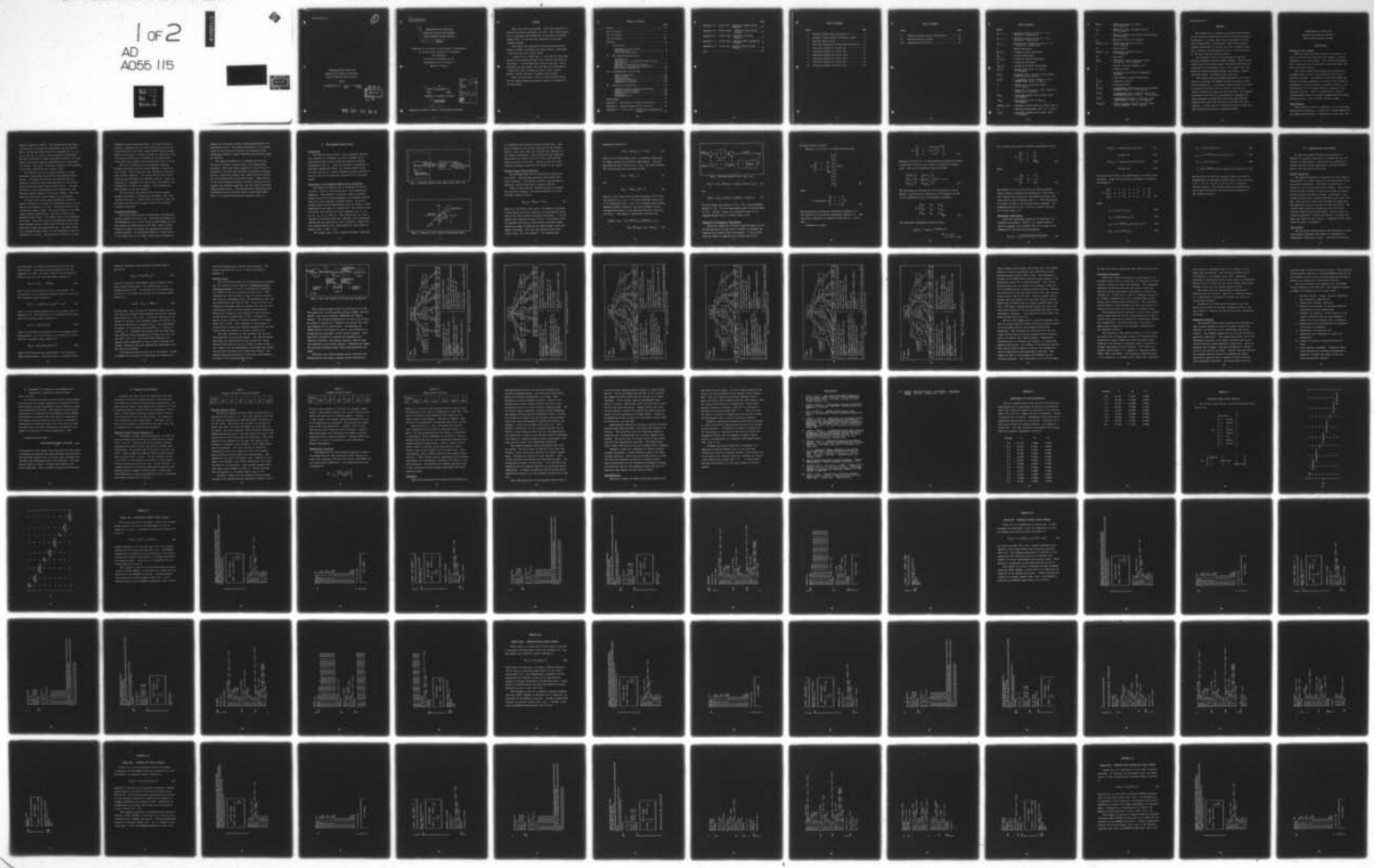
AD-A055 115 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 17/9
INVESTIGATION OF REAL-TIME SATELLITE-TO-SATELLITE TRACKING USIN--ETC(U)
MAR 78 H J LAUGHLIN

UNCLASSIFIED AFIT/GCS/EE/78-3 NL

1 OF 2
AD
A055 115



AFIT/GCS/EE/78-3

(1)

INVESTIGATION OF REAL-TIME SATELLITE-TO-SATELLITE TRACKING USING EXTENDED KALMAN FILTERS

THESIS

AFIT/GCS/EE/78-3

Henry J. Laughlin
Capt USAF

(14) AFIT/GCS/EE/78-3

(6) INVESTIGATION OF REAL-TIME SATELLITE-TO-SATELLITE TRACKING USING EXTENDED KALMAN FILTERS.

(9) Master's THESIS,

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

(10) by
Henry J. Laughlin, B.S.
Capt USAF
Graduate of Computer Systems

(11) March 1978

(12) 116 p.

Approved for public release; distribution unlimited

Preface

This report has two purposes. The first purpose is to satisfy the thesis requirement for AFIT. The second purpose is to investigate the feasibility of adapting an extended Kalman filter to the real-time satellite-to-satellite tracking problem.

This thesis was sponsored by the System Directorate branch of SAMSO, Los Angelos Air Force Station, California. I hope they find this thesis useful.

I would like to thank Capt. J. Gary Reid for his formulation of the extended Kalman filter and his assistance on the thesis. I would especially like to thank Mrs. Audrey Goldsmith for her support with the PDP-11/45 minicomputer. I would also like to thank my advisor, Capt. Peter E. Miller, for the latitude to complete this project.

Last, but not least, I wish to thank my wife, Karen, for her understanding and patience during the preparation of this thesis.

Table of Contents

List of Figures

List of Tables

List of Symbols

Symbol	
$S_{x,\ y,\ z}$	= Position estimate in the x, y, or z direction, respectively
$V_{x,\ y,\ z}$	= Velocity estimate in the x, y, or z direction, respectively
$A_{x,\ y,\ z}$	= Acceleration estimate in the x, y, or z direction, respectively
R	= Range measurement
$ZP_1(t_i^-)$	= Predicted range measurement
AZ	= Azimuth measurement
$ZP_2(t_i^-)$	= Predicted azimuth measurement
EL	= Elevation measurement
$ZP_3(t_i^-)$	= Predicted elevation measurement
$\bar{x}(t_i^-)$	= System state vector at time t_i (9 elements)
$\bar{x}(t_i^-)$	= A priori filter estimate of the system state vector at time t_i
$\bar{x}(t_i^+)$	= A posteriori filter estimate of the system state vector at time t_i
$\bar{\Phi}$	= System state transition matrix (9 X 9 matrix)
Φ	= Submatrix of the system state transition matrix (3 X 3 matrix)
$\bar{w}(t_i)$	= Zero mean Gaussian white noise vector (9 elements)
$\bar{z}(t_i)$	= Measurement vector at time t_i (3 elements)
$\bar{h}(\bar{x}(t_i), t_i))$	= Nonlinear vector function of $\bar{x}(t_i)$ and t_i
h	= Predicted measurement (ZP_1, ZP_2, or ZP_3)
$\bar{v}(t_i)$	= Zero mean Gaussian white noise vector (3 elements)

$\bar{P}(t_i)$	= Covariance matrix at time t_i (9 X 9 matrix)
P_{xy}	= Submatrix of the covariance matrix (3 X 3 matrix)
$\bar{Q}$	= Discrete driving noise matrix approximation (9 X 9 matrix)
Q	= Submatrix of $\bar{Q}$ (3 X 3 matrix)
$\bar{K}(\bar{x}(t_i^-), t_i)$	= Kalman gain matrix at time t_i (9 X 3 matrix)
K	= Column from $\bar{K}$
$\bar{H}(\bar{x}(t_i^-), t_i)$	= Measurement matrix at time t_i (3 X 9 matrix)
a^T	= Row from $\bar{H}$
$\bar{R}(t_i)$	= Measurement noise covariance matrix at time t_i (3 X 3 matrix)
r	= Element from the diagonal of $\bar{R}$
I	= Identity matrix
γ	= Acceleration correlation (bandwidth) parameter
Δ	= Time duration between measurements
$\delta(j,k)$	= Weighting function
NE	= Normalized Error
$\bar{S}(t_i^+)$	= A posteriori general square root covariance matrix at time t_i (9 X 9 matrix)
$\bar{U}(t_i^+)$	= A posteriori upper triangular square root covariance matrix at time t_i (9 X 9 matrix)
$\bar{D}(t_i^+)$	= A posteriori diagonal square root covariance vector at time t_i (9 elements)
$\bar{U}$-$\bar{D}(t_i^+)$	= Upper triangular matrix $\bar{U}$ with $\bar{D}$ stored on the diagonals (9 X 9 matrix)

Abstract

The feasibility of adapting an extended Kalman filter to the real-time satellite-to-satellite tracking problem is investigated. A filter using linear propagation and non-linear measurements is formulated. This filter accepts tracker measurements to provide the state estimates (position, velocity, and acceleration) of the tracked satellite in the local inertial reference frame.

Six different filters are programmed in American National Standards Institute (ANSI) FORTRAN. These filters employ various covariance update techniques and methods of processing the measurement vector. The programs are compared for computer program storage requirements, program execution speeds, and computational precision.

The computer program storage requirements are determined by utilizing the FORTRAN IV-PLUS compiler on a PDP-11/45 minicomputer. The program execution speeds are found by timing on the PDP-11/45 the filter's reaction to a typical rendezvous between two space vehicles. The computer precision of each filter is determined by comparing the state estimates obtained from the 60-bit word CDC-6600 computer with those from the 32-bit word PDP-11/45 minicomputer. A final filter is selected as best for the real-time satellite-to-satellite tracking problem.

INVESTIGATION OF REAL-TIME SATELLITE-TO-SATELLITE TRACKING USING EXTENDED KALMAN FILTERS

I. Introduction

Statement of the Problem

There are many situations where it is necessary to track one accelerating satellite from another. A space rendezvous is one such example. The tracking information that is required for this maneuver must be real-time (current) and highly accurate.

This thesis studies the feasibility of adapting an extended Kalman filter to the problem of satellite-to-satellite tracking. The problem of providing timely and accurate tracking information is solved in this thesis by determining the state estimates (position, velocity, and acceleration) of the tracked satellite relative to the tracking satellite. This is accomplished by using an extended Kalman filter of linear propagation and nonlinear measurements in a local inertial reference frame.

Scope/Approach

The problem is solved by combining a set of nonlinear Kalman filter equations, a collection of linear Kalman filter update mechanizations, a definition of real-time, and a

typical rendezvous scenario. The definition of real-time is being able to process the measurements (R, AZ, and EL) from a tracker and produce the state estimates (S_x, V_x, A_x, S_y, V_y, A_y, S_z, V_z, and A_z) while allowing a time margin for their use before the next measurements arrive. For this thesis, the tracker supplies measurements once every tenth of a second. The typical rendezvous scenario is the meeting of two space vehicles in low circular orbit.

The determination of the state estimates is accomplished by programming and analyzing six extended Kalman filters. The first five filters use modifications of Bierman's covariance update mechanizations of the linear state/linear measurement Kalman filter (Ref 2). The modifications to these mechanizations are documented by the software design technique of Structured Design. This technique will facilitate future changes to the programs.

The first five filters employ different covariance update techniques and process the measurement vector (R, AZ, and EL) one component at a time. The first filter is the conventional Kalman filter. The second filter is the stabilized (Joseph) Kalman filter. The third filter is the modified Potter filter. The fourth filter is the modified U-D filter. The fifth filter is the modified upper triangular (Carlson filter). These first five filters are optimized for storage and computation time. The sixth filter is a standard Kalman filter, but the measurement vector is processed all at once. The program for filter six is also

documented using Structured Design. The sixth filter is used as a reference for the first five filters and is considered the worst case from a computational load viewpoint. There is no attempt to optimize storage or computation time. Its filter equations are programmed in the pure matrix-vector form using general matrix subroutines (Ref 6 :65-66).

The filter algorithms are programmed in American National Standards Institute (ANSI) FORTRAN, as described in X.3-1966. This is done for easy transport of the source decks between computers. The source code for each filter algorithm was developed on a CDC-6600 computer. After the completion of debugging, the source codes (card decks) were transported to a PDP-11/45 computer. The programs were then executed on the minicomputer.

The six filters are compared for computer program storage requirements, program execution speeds, and computational precision. A final filter is selected as best for providing real-time and accurate satellite-to-satellite tracking information.

Contributions/Products

This thesis makes several contributions and generates various products for future use. The first contribution is the determination of a set of three extended Kalman filters that satisfy the definition of real-time. Each of these filters is capable of solving the satellite-to-satellite tracking problem. An interesting feature of these filters is the exploitation of the most current state estimates to

compute the measurement equation during processing of the measurement vector. The second contribution is the optimization of the filters for storage and computation time. Significant savings in memory locations and execution speed are obtained.

The third contribution is a comparison of the test filters for storage, execution time, and computational precision. Based upon these comparisons a best filter is selected for the real-time satellite-to-satellite tracking problem. The useful products that remain from this thesis are the documentations of the test filters. These consist of the filter program listings, structure diagrams for each program, the nonlinear equations, and the linear covariance update mechanizations. This documentation facilitates ease of modification for alternate target acceleration models (Ref 5) or alternate measurement equations (Ref 4).

II. The Extended Kalman Filter

Introduction

The extended Kalman filter can be a powerful and practical approach to estimating the state estimates of an accelerating space vehicle given the range and tracking measurements (Ref 9). Since this problem is inherently a nonlinear estimation problem, the extended Kalman filter has been found to be a useful compromise between estimation accuracy and ease of computation because of its recursive nature.

Description of the Extended Kalman Filter Application

The physical problem is how to determine the states (position, velocity, and acceleration) of an approaching satellite. This problem is solved by using an extended Kalman filter with linear propagation and nonlinear measurements in a local inertial coordinate frame. The filter accepts three observed measurements from the tracker. These measurements are range (R), azimuth (AZ), and elevation (EL). The filter then produces the state estimates (S_x, V_x, A_x, S_y, V_y, A_y, S_z, V_z, and A_z). The notations S_x, V_x, and A_x represent the relative position, velocity, and acceleration between the two satellites in the x direction, respectively. The extended Kalman filter application for this problem is shown in Fig. 1 (Ref 6:41).

The tracker uses a local inertial reference frame that

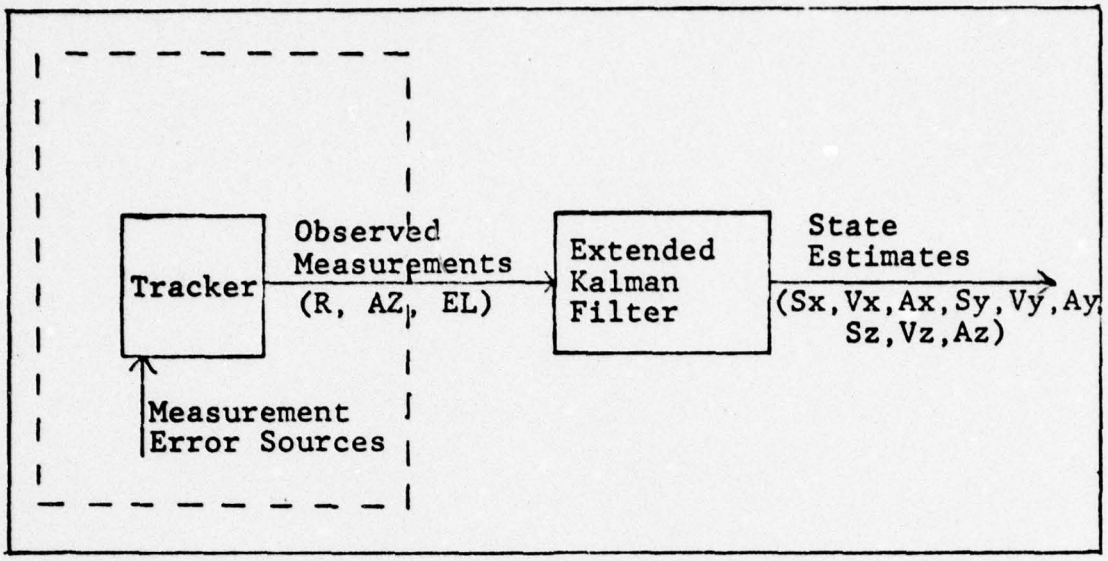


Fig. 1 Extended Kalman Filter Application (Ref 6:41)

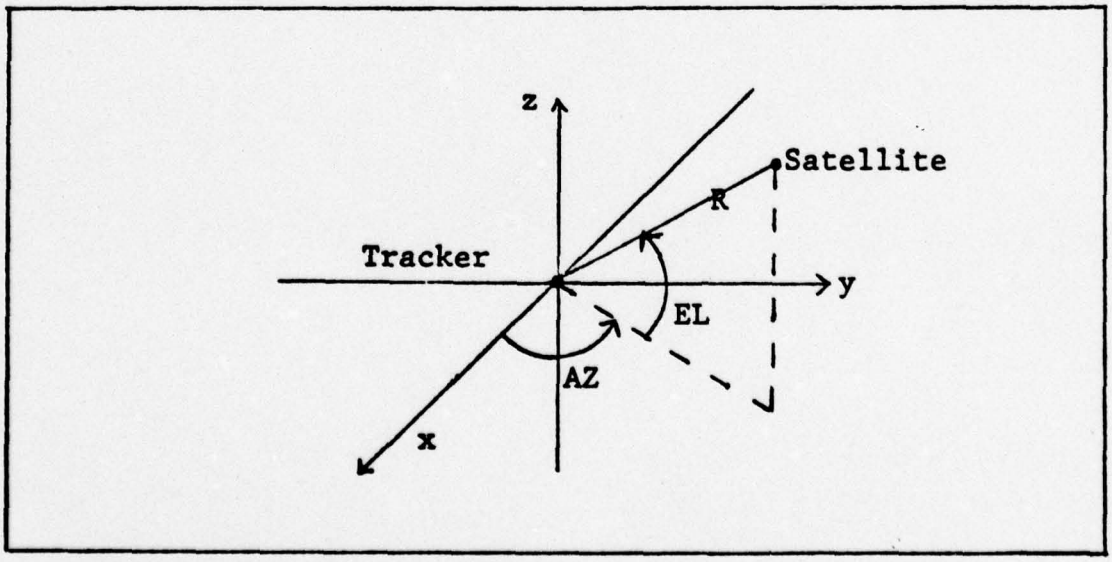


Fig. 2 Tracker's Local Inertial Coordinate Frame

is established upon initial satellite acquisition. The x axis is aligned in the initial direction to the tracked vehicle. The y and z axes are selected with reference to the tracker orientation. The selection of this reference frame forces the initial R to be on the x axis and the initial AZ and EL to be zero. Figure 2 shows the local inertial coordinate frame with its origin at the tracker.

Extended Kalman Filter Equations

The extended Kalman filter equations are stated for this thesis. The pertinent equations are divided into three categories: the system equations, the propagation equations, and the measurement equations (Ref 10).

Since a local inertial coordinate system is assumed for the state estimates, the state equation in the time invariant discrete form is

$$\bar{x}(t_{i+1}) = \bar{\Phi}\bar{x}(t_i) + \bar{w}(t_i) \qquad (1)$$

where $\bar{x}$ is the system state vector (9 elements) consisting of the relative position, velocity, and acceleration along each of the three inertial coordinate directions. $\bar{\Phi}$ is the system state transition matrix (9 X 9). The notation (9 X 9) specifies the size of a matrix where the first number equals the number of rows and the second number equals the number of columns. The zero mean Gaussian white noise vector, $\bar{w}(t_i)$, has nine elements. The discrete time

measurement equation is

$$\bar{z}(t_i) = h(\bar{x}(t_i), t_i) + \bar{v}(t_i) \tag{2}$$

where $\bar{z}$ is the measurement vector (3 elements) consisting of range, azimuth, and elevation measurements. The zero mean Gaussian white noise vector, $\bar{v}(t_i)$, has three elements. The filter propagation equations are then

$$\bar{x}(t_i^-) = \bar{\Phi}\bar{x}(t_{i-1}^+) \tag{3}$$

and

$$\bar{P}(t_i^-) = \bar{\Phi}\bar{P}(t_{i-1}^+)\bar{\Phi}^T + \bar{Q} \tag{4}$$

where $\bar{\Phi}$ is the system state transition matrix (9 X 9) over the interval (t_{i-1}^+, t_i^-). $\bar{P}$ is the covariance matrix (9 X 9) of estimation error. $\bar{Q}$ is a 9 X 9 diagonal matrix whose diagonal elements consist of the noise strength of each corresponding element of the zero mean Gaussian noise process $\bar{w}(t_i)$. The update or measurement equations are

$$\bar{K}(\bar{x}(t_i^-), t_i) = \bar{P}(t_i^-)\bar{H}^T(\bar{x}(t_i^-), t_i)\left[\bar{H}(\bar{x}(t_i^-), t_i) \cdot \bar{P}(t_i^-)\bar{H}^T(\bar{x}(t_i^-), t_i) + \bar{R}(t_i)\right]^{-1} \tag{5}$$

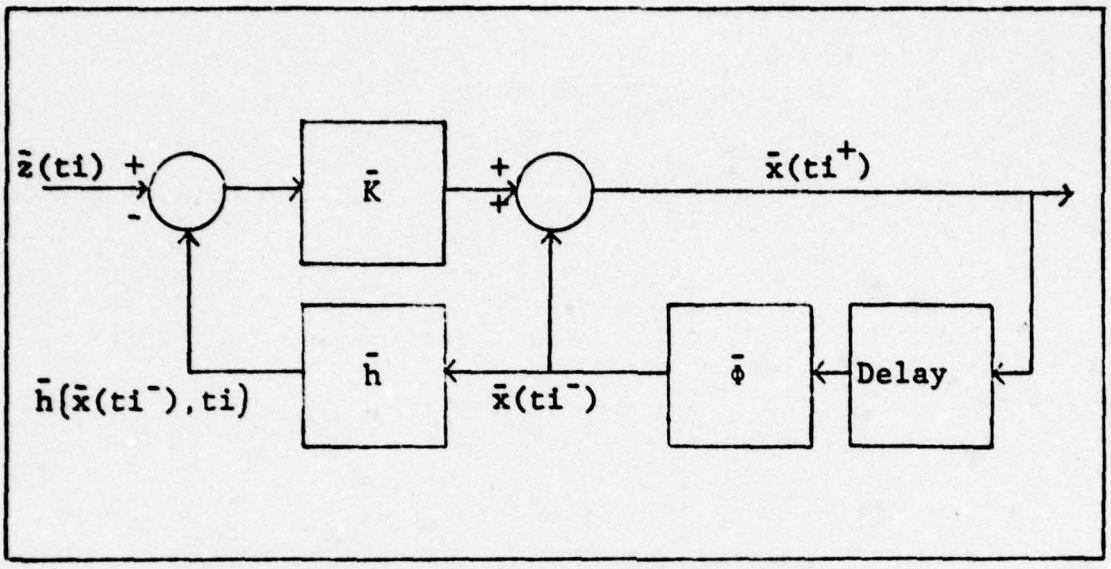


Fig. 3 Extended Kalman Filter (Ref 6:63)

$$\bar{x}(t_i^+) = \bar{x}(t_i^-)+\bar{K}(\bar{x}(t_i^-),t_i)\left[\bar{z}(t_i)-\bar{h}(\bar{x}(t_i^-),t_i)\right] \quad (6)$$

and

$$\bar{P}(t_i^+) = \bar{P}(t_i^-)-\bar{K}(\bar{x}(t_i^-),t_i)\bar{H}(\bar{x}(t_i^-),t_i)\bar{P}(t_i^-) \quad (7)$$

$\bar{K}$ is the Kalman gain matrix (9 X 3). $\bar{H}$ is the measurement matrix (3 X 9). $\bar{R}$ is the measurement noise covariance matrix (3 X 3). As Fig. 3 shows, the recursive nature of the extended Kalman filter is obvious (Ref 6 :63).

Reduction of Propagation Computations

Since the comparison of filter execution speeds is one of the objectives of this thesis, methods of reducing the computational workload were investigated. It was noticed that the number of computations involving the $\bar{P}$ and $\bar{\Phi}$

matrices could be reduced.

Equation 1 is of the 3 X 3 block decoupled from

$$\bar{x} = \begin{bmatrix} \Phi & 0 & 0 \\ 0 & \Phi & 0 \\ 0 & 0 & \Phi \end{bmatrix}_{9X9} \begin{bmatrix} S_x \\ V_x \\ A_x \\ S_y \\ V_y \\ A_y \\ S_z \\ V_z \\ A_z \end{bmatrix} + \bar{w}(t_i) \tag{8}$$

where

$$\Phi = \text{exponential} \left(\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1/\gamma \end{bmatrix} \Delta \right) \tag{9}$$

The acceleration correlation (bandwidth) parameter is γ. The duration of time between measurement updates is Δ. Note that these equations are decoupled around the x, y, and z axes.

Solving for Φ gives

$$\Phi = \begin{bmatrix} 1 & \Delta & -\gamma^2+\gamma\Delta+\gamma^2 e^{-\Delta/\gamma} \\ 0 & 1 & \gamma(1-e^{-\Delta/\gamma}) \\ 0 & 0 & e^{-\Delta/\gamma} \end{bmatrix} \tag{10}$$

Applying the value of Φ to the propagation equations produces a substantial saving in the number of propagation computations. The state propagation equation becomes

$$\begin{bmatrix} S_j(t_i^-) \\ V_j(t_i^-) \\ A_j(t_i^-) \end{bmatrix} = \Phi \begin{bmatrix} S_j(t_{i-1}^+) \\ V_j(t_{i-1}^+) \\ A_j(t_{i-1}^+) \end{bmatrix} \text{ for } j = x,y,z \tag{11}$$

The big saving is achieved with the propagation of the $\bar{P}$ matrix. The $\bar{P}$ matrix is considered to be composed of nine (3 X 3) submatrices that are propagated separately.

$$\bar{P} = \begin{bmatrix} P_{xx} & P_{xy} & P_{xy} \\ P_{yx} & P_{yy} & P_{yz} \\ P_{zx} & P_{zy} & P_{zz} \end{bmatrix}_{9X9} \tag{12}$$

The covariance propagation equation becomes

$$P_{jk}(t_i^-) = \Phi P_{jk}(t_{i-1}^+)\phi^T + Q\delta(j,k)$$

$$\text{for } j = x,y,z \quad k = x,y,z \tag{13}$$

Q is a white driving noise submatrix approximation for $\bar{Q}$.

$$\bar{Q} = \begin{bmatrix} Q & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & Q \end{bmatrix}_{9X9} \qquad (14)$$

where

$$Q = \begin{bmatrix} QX & 0 & 0 \\ 0 & QV & 0 \\ 0 & 0 & QA \end{bmatrix} \qquad (15)$$

The elements of Q are selected by the design engineer during tuning to optimize the performance of the filter. A more complex approximation of the discrete driving noise term could be used if desired (Ref 1). This modification can easily be made to the existing filter programs. The weighting function $\delta(j,k)$ is one if j equals k and zero if j is not equal to k.

Measurement Computations

Since the measurement equations are nonlinear, the estimated measurements, $\bar{h}(\bar{x}(t_i^-),t_i)$, depend upon the current a priori state estimates This can be seen in the equations for the predicted measurements.

$$ZP_1(t_i^-) = \sqrt{x_1(t_i^-)^2+x_4(t_i^-)^2+x_7(t_i^-)^2} \qquad (16)$$

$$ZP_2(t_i^-) = \arctan\left(x_4(t_i^-)/x_1(t_i^-)\right) \tag{17}$$

$$= \arctan(u) \tag{18}$$

$$ZP_3(t_i^-) = \arcsin\left(x_7(t_i^-)/ZP_1(t_i^-)\right) \tag{19}$$

$$= \arcsin(v) \tag{20}$$

The linearized $\bar{H}$ matrix also depends upon the current state estimates. Using the conventions established in Equations 16 through 20,

$$\bar{H} = \begin{bmatrix} H_{11} & 0 & 0 & H_{14} & 0 & 0 & H_{17} & 0 & 0 \\ H_{21} & 0 & 0 & H_{24} & 0 & 0 & 0 & 0 & 0 \\ H_{31} & 0 & 0 & H_{34} & 0 & 0 & H_{37} & 0 & 0 \end{bmatrix} \tag{21}$$

where

$$H_{11} = x_1(t_i^-)/ZP_1(t_i^-) \tag{22}$$

$$H_{21} = -u/\left((1+u^2)x_1(t_i^-)\right) \tag{23}$$

$$H_{31} = \left(-v/\sqrt{1-v^2}\right)\left(x_1(t_i^-)/ZP_1(t_i^-)^2\right) \tag{24}$$

$$H_{14} = x_4(t_i^-)/ZP_1(t_i^-) \tag{25}$$

$$H_{24} = 1/\left((1+u^2)x_1(t_i^-)\right) \tag{26}$$

$$H_{34} = \left(-v/\sqrt{1-v^2}\right)\left(x_4(t_i^-)/ZP_1(t_i^-)^2\right) \tag{27}$$

$$H_{17} = x_7(t_i^-)/ZP_1(t_i^-) \tag{28}$$

$$H_{37} = \left[\left(1/\sqrt{1-v^2}\right)\left(1/ZP_1(t_i^-)\right)\right] - \left[vx_7(t_i^-)/ZP_1(t_i^-)^2\right] \tag{29}$$

The satellite-to-satellite tracking problem is solved by implementing the developed extended Kalman filter equations 5, 6, 7, 11, 13, 16, 18, 20, and 21 into a recursive computer program. The program accepts measurements from the tracker and produces the desired state estimates of the tracked satellite.

III. Implementation and Testing

To test the extended Kalman filter equations of Chapter II, a typical rendezvous is proposed and six test filters are investigated. The filters are analyzed for computer program storage requirements, program execution speeds, and computational precision.

Typical Rendezvous

The typical rendezvous is assumed to be the tracker's observations of a tracked satellite for ten seconds following initial acquisition. The tracker is mounted on a vehicle in low circular orbit (193 X 193 km). The notation (193 X 193 km) specifies an orbit with perigee and apogee at 193 km above the earth's surface, respectively. The tracked vehicle is in a higher elliptical orbit (193 X 820 km). Both satellite orbits are in nearly the same plane and rendezvous occurs at perigee for both satellites. The tracker begins providing information when the scanned satellite comes within 55,000 meters. From the point of initial acquisition, tracking information is provided ten times per second for ten seconds. A sampling of this information is shown in Appendix A.

Test Filters

The six filter implementations are variations of covariance update techniques and methods of processing the measurement (observation) vector. The first five filters

use adaptations of Bierman's mechanizations for the non-linear system. They process the measurement vector one component at a time. The first filter is the conventional Kalman filter with the covariance update equation of

$$\bar{P}(t_i^+) = \bar{P}(t_i^-) - Ka^T\bar{P}(t_i^-) \tag{30}$$

where K is a column from $\bar{K}$ and a^T is a row from $\bar{H}$. The second filter is the stabilized Kalman (Joseph) filter with the covariance update equation of

$$\bar{P}(t_i^+) = (I-Ka^T)\bar{P}(t_i^-)(I-Ka^T)^T + KrK^T \tag{31}$$

where I is the identity matrix and r is an element from the diagonal of $\bar{R}$. The third filter is a modified Potter filter with the covariance update equation of

$$\bar{P}(t_i^+) = \bar{S}(t_i^+)\bar{S}(t_i^+)^T \tag{32}$$

where $\bar{S}$ is obtained from Bierman's Potter covariance update mechanization. The fourth filter is a modified U-D filter with the covariance update equation of

$$\bar{P}(t_i^+) = \bar{U}(t_i^+)\bar{D}(t_i^+)\bar{U}(t_i^+)^T \tag{33}$$

where $\bar{U}$ and $\bar{D}$ are obtained from Bierman's U-D covariance update mechanization. The fifth filter is a modified upper

triangular (Carlson) filter with the covariance update equation of

$$\bar{P}(t_i^+) = \bar{U}(t_i^+)\bar{U}(t_i^+)^T \tag{34}$$

where $\bar{U}$ is obtained from Bierman's upper triangular covariance update mechanization. The sixth filter is the standard Kalman filter. It processes all the observations ($\bar{z}$ vector) at the same time and uses the covariance update equation of

$$\bar{P}(t_i^+) = \bar{P}(t_i^-) - \bar{K}\bar{H}\bar{P}(t_i^-) \tag{35}$$

Filters three, four, and five are modified square root filters and should produce state estimates with better accuracy than filters one, two, or six. For faster execution at the expense of accuracy, these modified square root filters use the Matrix RSS method to compute $\bar{P}(t_{i+1}^-)$ and then generate the square root covariance matrices ($\bar{S}(t_{i+1}^-)$, $\bar{U}$-$\bar{D}(t_{i+1}^-)$, and $\bar{U}(t_{i+1}^-)$). Therefore they possess only the same numerical precision during propagation as filters one, two, and six (Ref 7:Ch 7,16). The sixth filter represents the typical filter implementation and is used as reference for storage, execution time, and computational comparisons with the first five filters.

No attempt was made to tune any of the filters. Tuning is defined as the process of optimizing the parameters of

the $\bar{Q}$ and $\bar{R}$ matrices for optimal filter accuracy. The initial conditions for $\bar{x}_0$, $\bar{P}_0$, $\bar{Q}$, and $\bar{R}$ are shown in Appendix B.

Software Design

The six extended Kalman filter algorithms are developed with the software design technique of Structured Design. "Structured design is the art of designing the components of a system in the best possible way (Ref 11:7)." This technique was chosen because it tends to produce programs that have low development costs, low maintenance costs, and low modification costs (Ref 11:254). These low costs are achieved by designing computer programs that have low coupling between modules and high cohesion within modules. Coupling is a measure of the strength of interconnection between software modules (Ref 11:116). Cohesion is a measure of the strength of internal bonding of a software module (Ref 11:143). This technique is used because it produces more flexible and complete documentation than flow charts, which are limited to control coupling only.

The major components of this technique are the data flow graph and the structure diagram. The data flow graph identifies the primary flow of data through the system. The data flow graph for all six filters is shown in Fig. 4. This figure shows the measurements as the high level inputs and the state estimates as the high level outputs. The primary processing functions are the update and propagate portions of the extended Kalman filter.

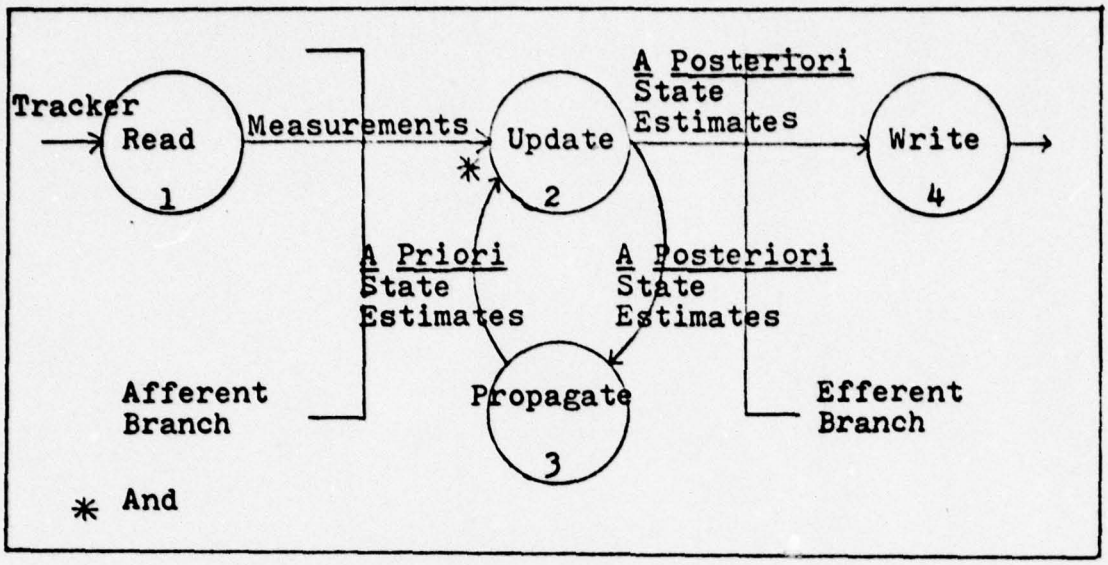
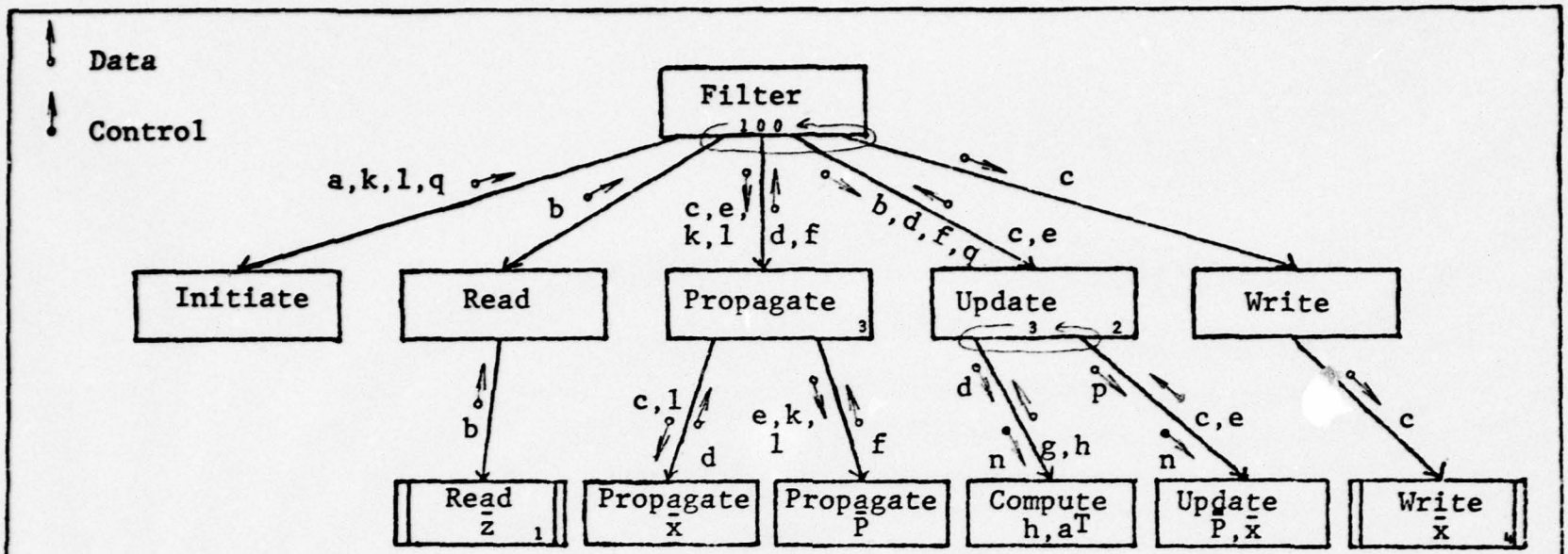


Fig. 4 Data Flow Diagram for Filters One Through Six

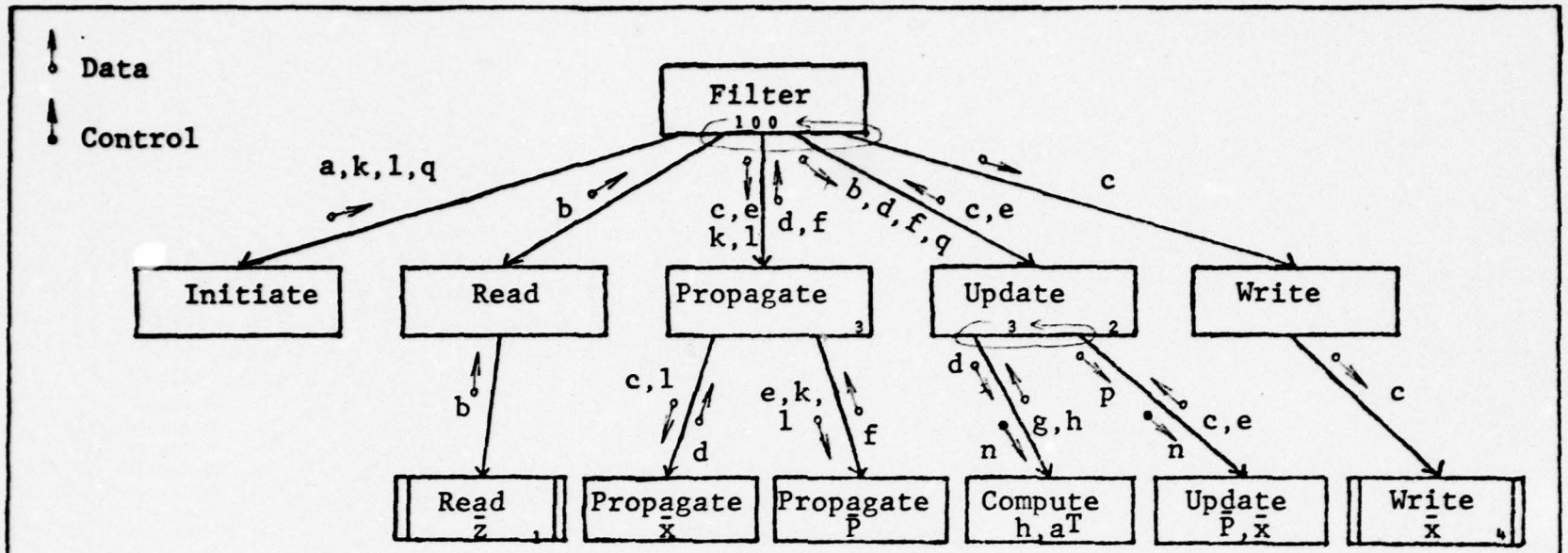
The structure diagram shows the factoring of the data flow graph into its corresponding afferent (READ), efferent (WRITE), and central transform (PROPAGATE and UPDATE) modules. The structure diagrams for the first five filters are shown in Figures 5-9. Figure 10 illustrates the structure diagram for the sixth filter. By comparing the UPDATE modules from Figures 5-9 and 10, the difference in processing the measurement vector becomes apparent. Since each filter program will probably be implemented as a separate subroutine, some design license is used to name the submodules of the filter program. Submodules are named for functions performed rather than using subroutine call names.

The first five filter programs can be criticized for having modules with higher coupling between modules and



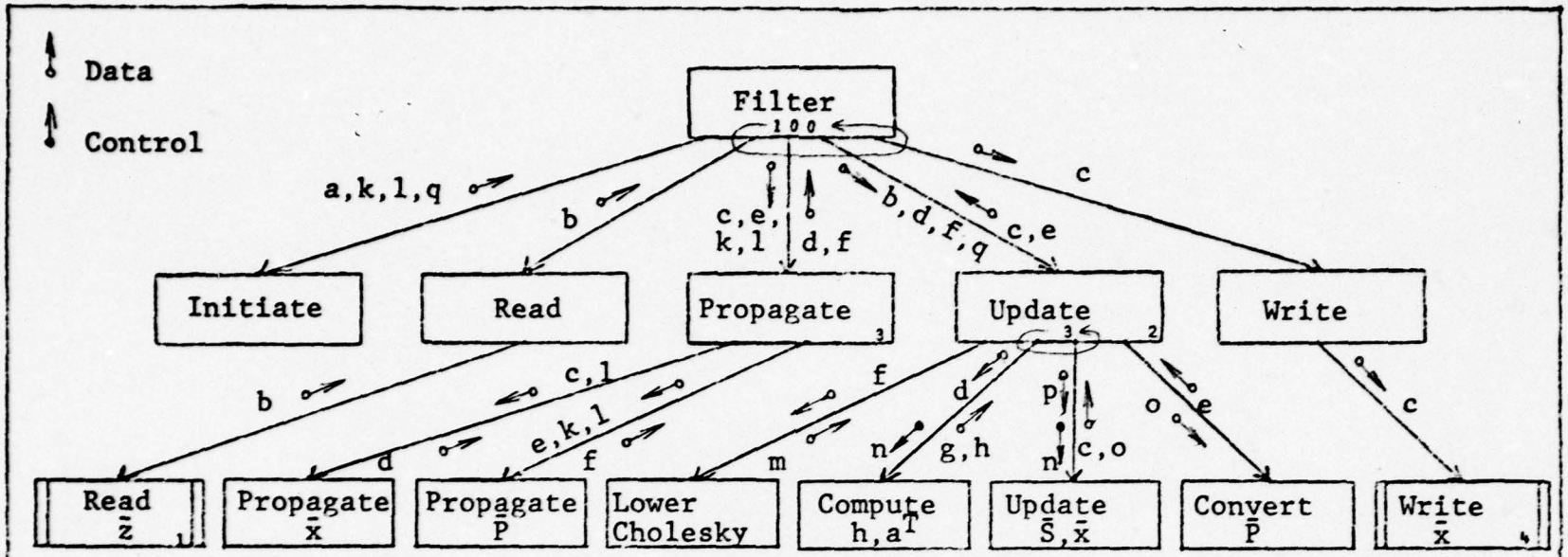
a. Initial filter matrices $(\bar{x}_0, \bar{P}_0)$
b. Measurement vector $(\bar{z}(t_i))$
c. A posteriori state estimate vector $(\bar{x}(t_{i-1}^+))$
d. A priori state estimate vector $(\bar{x}(t_i^-))$
e. A posteriori covariance matrix $(\bar{P}(t_{i-1}^+))$
f. A priori covariance matrix $(\bar{P}(t_i^-))$
g. Predicted measurement element $(h(\bar{x}(t_i^-), t_i))$
h. Measurement row $(a(\bar{x}(t_i^-), t_i))$
j. Measurement noise covariance element (R(n))
k. Q approximation (QX, QV, QA)
l. State transition matrix (Φ)
n. Iteration number
p. b, d, f, g, h, j
q. Measurement noise covariance vector $(\bar{R}(t))$

Fig. 5 Structure Diagram for Filter One



a. Initial filter matrices $(\bar{x}_0, \bar{P}_0)$
b. Measurement vector $(\bar{z}(t_i))$
c. A posteriori state estimate vector $(\bar{x}(t_{i-1}^+))$
d. A priori state estimate vector $(\bar{x}(t_i^-))$
e. A posteriori covariance matrix $(\bar{P}(t_{i-1}^+))$
f. A priori covariance matrix $(\bar{P}(t_i^-))$
g. Predicted measurement element $(h(\bar{x}(t_i^-), t_i))$
h. Measurement row $(a(\bar{x}(t_i^-), t_i))$
j. Measurement noise covariance element $(R(n))$
k. Q approximation (QX, QV, QA)
l. State transition matrix (Φ)
n. Iteration number
p. b, d, f, g, h, j
q. Measurement noise covariance vector $(\bar{R}(t))$

Fig. 6 Structure Diagram for Filter Two



a. Initial filter matrices $(\bar{x}_0, \bar{P}_0)$

b. Measurement vector $(\bar{z}(t_i))$

c. A posteriori state estimate vector $(\bar{x}(t_{i-1}^+))$

d. A priori state estimate vector $(\bar{x}(t_i^-))$

e. A posteriori covariance matrix $(\bar{P}(t_{i-1}^+))$

f. A priori covariance matrix $(\bar{P}(t_i^-))$

g. Predicted measurement element $(h(\bar{x}(t_i^-), t_i))$

h. Measurement row $(a(\bar{x}(t_i^-), t_i))$

j. Measurement noise covariance element $(R(n))$

k. Q approximation (QX, QV, QA)

l. State transition matrix (Φ)

m. A priori lower triangular matrix $(\bar{S}(t_i^-))$

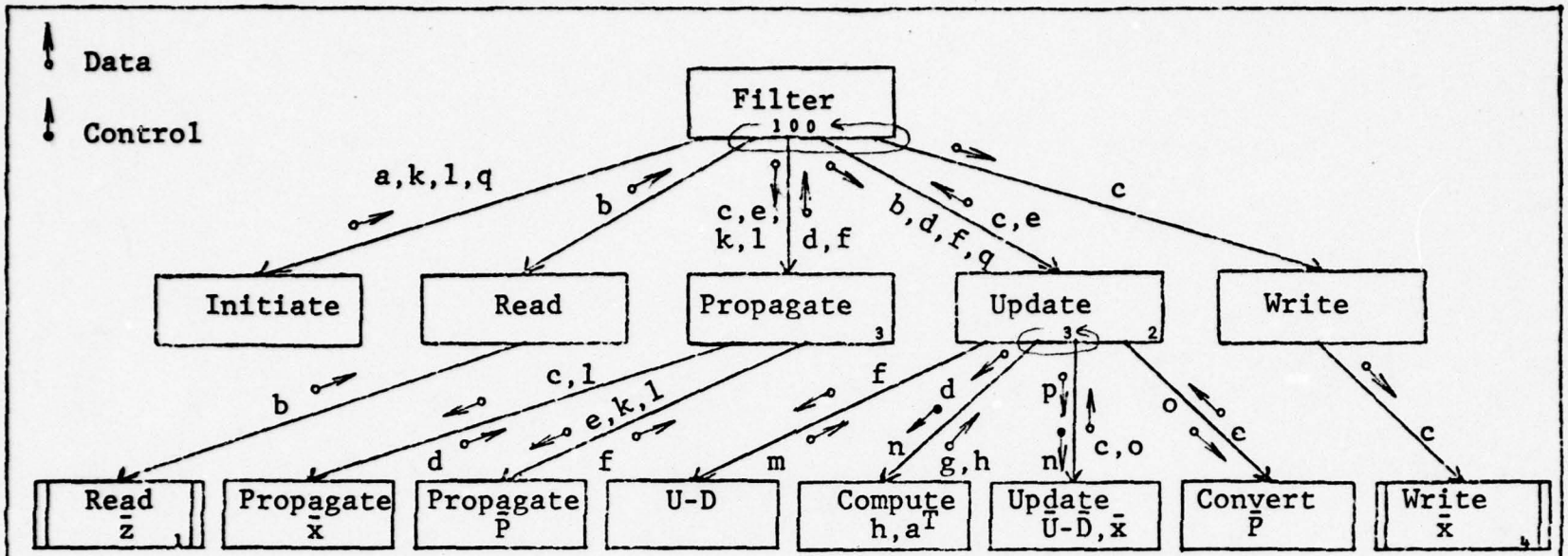
n. Iteration number

o. A posteriori square root matrix $(\bar{S}(t_i^+))$

p. b, d, g, h, j, m

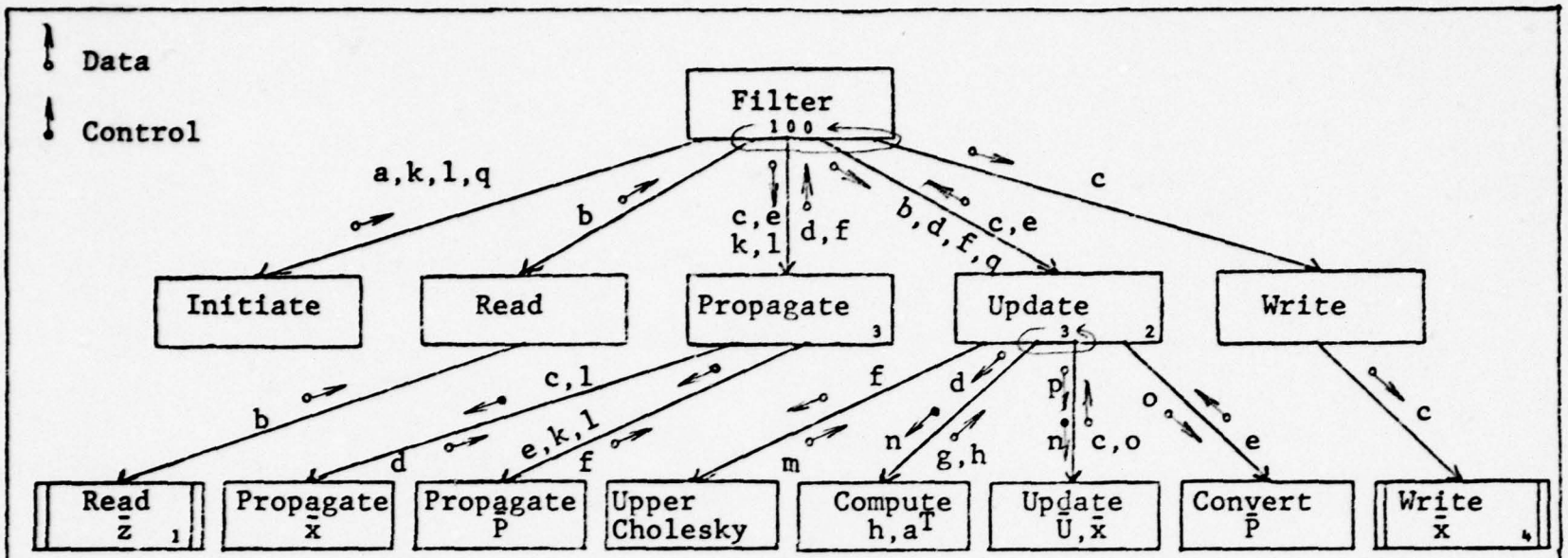
q. Measurement noise covariance vector $(\bar{R}(t))$

Fig. 7 Structure Diagram for Filter Three



a. Initial filter matrices $(\bar{x}_0, \bar{P}_0)$
b. Measurement vector $(\bar{z}(t_i))$
c. A posteriori state estimate vector $(\bar{x}(t_{i-1}^+))$
d. A priori state estimate vector $(\bar{x}(t_i^-))$
e. A posteriori covariance matrix $(\bar{P}(t_{i-1}^+))$
f. A priori covariance matrix $(\bar{P}(t_i^-))$
g. Predicted measurement element $(h(\bar{x}(t_i^-), t_i))$
h. Measurement row $(a(\bar{x}(t_i^-), t_i))$
j. Measurement noise covariance element (R(n))
k. Q approximation (QX, QV, QA)
l. State transition matrix (Φ)
m. A priori upper triangular matrix $(\bar{U}\text{-}\bar{D}(t_i^-))$
n. Iteration number
o. A posteriori upper triangular matrix $(\bar{U}\text{-}\bar{D}(t_i^+))$
p. b, d, g, h, j, m
q. Measurement noise covariance vector $(\bar{R}(t))$

Fig. 8 Structure Diagram for Filter Four



a. Initial filter matrices $(\bar{x}_0, \bar{P}_0)$

b. Measurement vector $(\bar{z}(t_i))$

c. A posteriori state estimate vector $(\bar{x}(t_{i-1}^+))$

d. A priori state estimate vector $(\bar{x}(t_i^-))$

e. A posteriori covariance matrix $(\bar{P}(t_{i-1}^+))$

f. A priori covariance matrix $(\bar{P}(t_i^-))$

g. Predicted measurement element $(h(\bar{x}(t_i^-), t_i))$

h. Measurement row $(a(\bar{x}(t_i^-), t_i))$

j. Measurement noise covariance element (R(n))

k. Q approximation (QX, QV, QA)

l. State transition matrix (Φ)

m. A priori upper triangular matrix $(\bar{U}(t_i^-))$

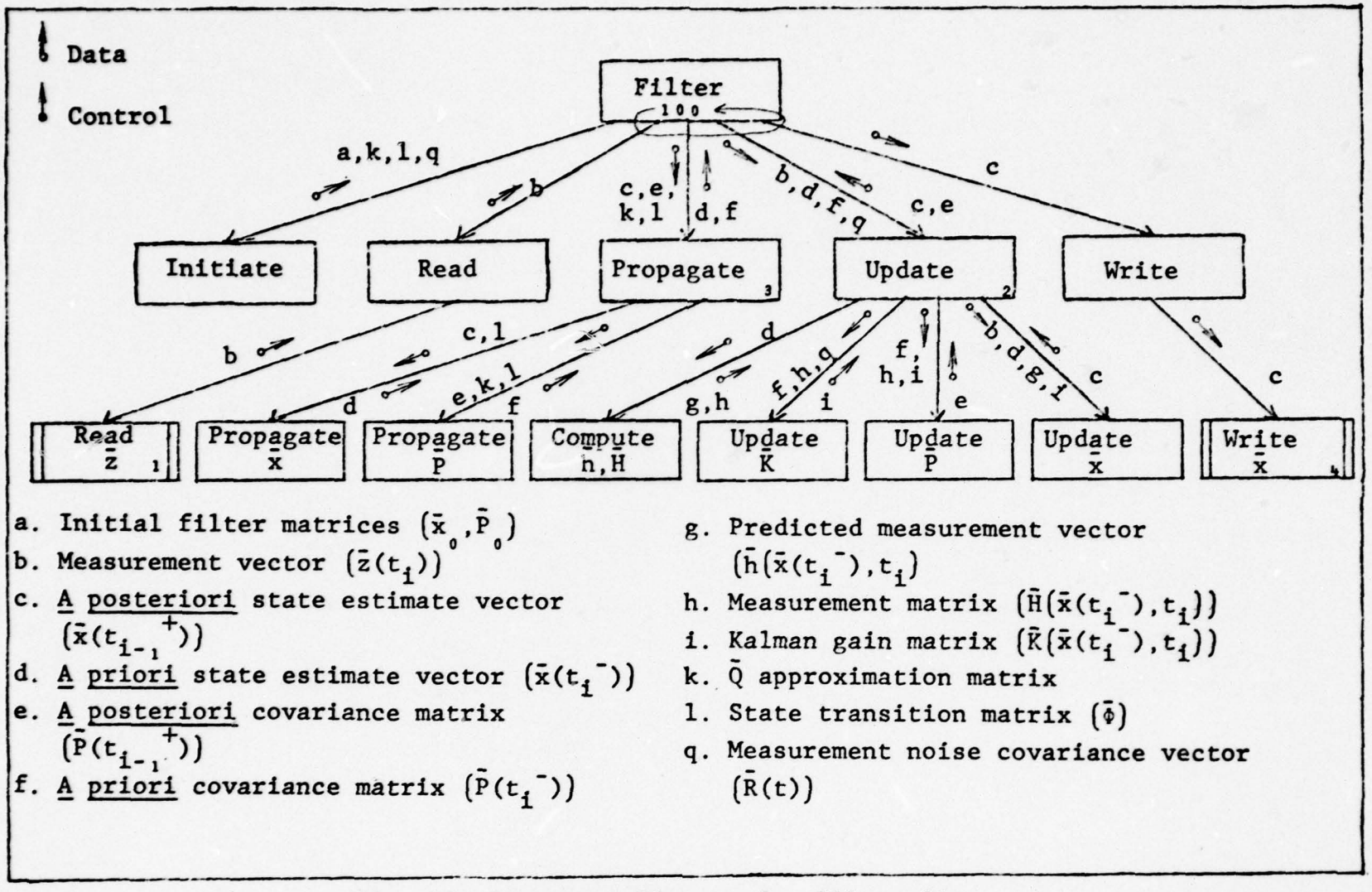
n. Iteration number

o. A posteriori upper triangular matrix $(\bar{U}(t_i^+))$

p. b, d, g, h, j, m

q. Measurement noise covariance vector $(\bar{R}(t))$

Fig. 9 Structure Diagram for Filter Five



a. Initial filter matrices $(\bar{x}_0, \bar{P}_0)$

b. Measurement vector $(\bar{z}(t_i))$

c. A posteriori state estimate vector $(\bar{x}(t_{i-1}^+))$

d. A priori state estimate vector $(\bar{x}(t_i^-))$

e. A posteriori covariance matrix $(\bar{P}(t_{i-1}^+))$

f. A priori covariance matrix $(\bar{P}(t_i^-))$

g. Predicted measurement vector $(\bar{h}(\bar{x}(t_i^-), t_i)$

h. Measurement matrix $(\bar{H}(\bar{x}(t_i^-), t_i))$

i. Kalman gain matrix $(\bar{K}(\bar{x}(t_i^-), t_i))$

k. $\bar{Q}$ approximation matrix

l. State transition matrix $(\bar{\Phi})$

q. Measurement noise covariance vector $(\bar{R}(t))$

Fig. 10 Structure Diagram for Filter Six

lower cohesion within modules than filter six. The UPDATE modules in filters one through five suffer from control coupling between modules (Ref 11:125). The subordinate modules of the UPDATE module must know which element of the measurement vector ($\bar{z}$) is being processed. This control coupling along with the normal data coupling is essential for the UPDATE modules to perform their functions. Modules that perform more than one function also have lower cohesion within modules (Ref 11:171-172). Ideally a module should perform only one function, but when more than one are performed then the module has low logical cohesion (Refll:153-154). The higher coupling between modules and lower cohesion within modules in filters one through five should make development, maintenance, and modification costs higher for these filters than filter six.

The data flow diagram and the structure diagrams along with the actual program listing constitute the major portion of the documentation for the extended Kalman filter algorithms. This documentation illustrates how the nonlinear system equations and the modified linear mechanizations are combined into useful programs. Because this documentation is so flexible and complete, it allows easy program modification. For example, it was noticed that the most current state estimates in filters one through five could be exploited in the measurement equations. This change was made to the programs and documented on the structure diagrams. This improvement to the filters might

not have been seen if flowcharts were used as documentation.

Programming Philosophy

Since this thesis investigates the application of an extended Kalman filter in real-time, the reduction of filter execution speeds is a major consideration. The programming philosophy minimizing execution time without incurring excessive storage requirements was applied to filters one through five. Filter six is used as a reference because it is a common implementation of the extended Kalman filter using general purpose matrix subroutines. This comparison should indicate if the optimization effort for filters one through five was worth the reduction in execution times.

The implementation of equations 11 and 13 saves considerable computation time during the propagation of $\bar{x}$ and $\bar{P}$. Further time reductions are achieved by eliminating unnecessary computations. The reductions are of two classes: sparse matrix computations and redundant computations of the symmetric $\bar{P}$ matrix.

The realization that several matrices (Φ, $\bar{R}$, $\bar{H}$, and Q) are sparse and the subsequent reduction of noncontributing computations greatly diminished filter execution speeds. Along with the decrease in execution times, a reduction in storage requirements is also achieved. The Φ matrix is further reduced from a 3 X 3 matrix to 4 constants (PHI12, PHI13, PHI23, and PHI33). The $\bar{R}$ matrix is decreased from a 3 X 3 matrix to a 3 element vector (R(1), R(2), and R(3)).

The Q matrix is diminished from a 3 X 3 matrix to 3 constants (QX, QV, and QA). The $\bar{H}$ matrix is reduced from a 3 X 9 matrix to a 9 element vector (a^T). Additional unnecessary computations during filter update are eliminated because as Eq. 21 shows only the first, fourth, and seventh elements of any row ever contain nonzero entries.

The second class of computational reductions are achieved with the $\bar{P}$ matrix. Since $\bar{P}$ is a symmetric matrix, it is unnecessary to recompute an element P_{ij} if P_{ji} has already been computed.

Because all the information contained in the full matrix can be found in the upper triangular portion, only that portion is computed and used during filter propagation and update.

Comparison Procedure

The extended Kalman filter programs are programmed in ANSI standard FORTRAN for ease of transport between the CDC-6600, the PDP-11/45 and an undetermined application computer. The FORTRAN programs for the six test filters are provided in Appendix C. The programs were built on the CDC-6600 machine because it was readily available and it possessed extensive program debugging capabilities. After the programs were debugged, they were transferred to the PDP-11/45 in the form of card decks. The PDP-11/45 was selected for program execution because it resembles the typical space-borne computer that is capable of 32-bit floating point arithmetic operations. The source decks were then

executed using the RSX-11M operating system. This operating system normally operates in a multiprogramming mode, but it was assumed to provide good execution speed approximations because the programs were individually executed.

The filter programs were compiled with the FORTRAN IV-PLUS compiler on the PDP-11/45. This compiler performs a number of optimizations on the source code.

1. Constant folding. Integer constant expressions are evaluated at compile-time.
2. Compile-time constant conversion.
3. Compile-time evaluation of constant subscript expressions in array calculations.
4. Argument-list merging. If two function or subroutine references have the same arguments, a single copy of the argument list is generated.
5. Branch instruction optimizations for arithmetic and logical IF statements.
6. Elimination of unreachable code.
7. Recognition and replacement of common subexpressions.
8. Removal of invariant computations from DO loops.
9. Local register assignment. Frequently referenced variables are retained (if possbile) in registers to reduce the number of load and store instructions required.

10. Assignment of frequently used variables and expressions to registers across DO loops.

(Ref 3 :Ch 4,4).

The six filter programs are tested with the one hundred data sets from the typical rendezvous and compared for computer program storage requirements, program execution speeds and computational precision. The computer program storage requirements are obtained from the Program Section Summary. This summary is provided after each PDP-11/45 compilation. The program execution speed for each filter is obtained by subtracting its input and output (I/O) time from its total execution time and then by dividing the difference by the number of iterations the filter accomplished. Thus

$$\text{program execution speed} = \frac{\text{total execution time - I/O time}}{100} \quad (36)$$

An evaluation of the computational precision for each filter is obtained by comparing the state estimates from the 60-bit word CDC-6600 computer with the 32-bit state estimates from the PDP-11/45 minicomputer after 100 iterations. This comparison shows the effect of finite word length on the filter algorithm. This is typically called round-off error.

IV. Results and Conclusion

To select the best filter for solving the real-time satellite-to-satellite tracking problem, the six filters are compared for computer program storage requirements, program execution speeds, and computational precision. All the filters performed as expected with the exception of filter three. Filter three continues to produce state estimates that vary greatly from the other filters. The reason for this condition is still undetermined. Therefore, filter three is not seriously considered for the best filter, but its comparisons are performed and documented.

Computer Program Storage Requirements

The computer program storage requirement for each filter is obtained from the PDP-11/45 Program Section Summary (Ref 3:Ch 1,8). Since the PDP-11/45 assembly language includes both single (16-bits) and double (32-bits) word instructions, the standard unit of storage is the byte (8-bits). Table I shows the computer program storage requirement for each filter in bytes. For the optimized filters 1-5, filter 1 is the smallest and filter 4 is the largest. Filter 6 proves to be the worst case - as was expected. It requires from 87 to 50 percent additional storage over the smallest and the largest optimized filter, respectively. This difference in storage requirements is due to the associated matrix subroutines of filter 6.

Table I

Computer Program Storage Requirements

Filter	1	2	3	4	5	6
Bytes	1538	1763	1843	1916	1874	2890

Program Execution Speeds

The average program execution speed for each filter is obtained by utilizing Eq. 36. The total execution and total I/O times are determined by using the SECNDS instruction on the PDP-11/45. This instruction provides the central processor unit (CPU) time from the moment it is set until it is called. The total execution time is found by setting the SECNDS instruction prior to the main program iteration. The instruction is then called after 100 iterations. This total execution time contains all the times for computer logic, indexing of variables, data transfers, filter computations, and I/O operations. The total I/O time is derived by removing all the filter computations from the program's main iteration and then running the program again. This removes all the times for computer logic, indexing of variables, data transfers, and filter computations; but leaves the time for I/O operations. These average program execution speed results appear in Table II. Actual times may vary +15 percent to -10 percent (Ref 8 :App B, 10).

As Table II shows, all the optimized filters execute in about 0.05 seconds with the exception of filters 3 and 5.

Table II

Program Execution Speeds

Filter	1	2	3	4	5	6
Sec	.0496	.0502	.1011	.0503	.1239	.0763

Filters 3 and 5 execute in 0.10 and 0.12 seconds, respectively. This slower execution rate is due to the need for taking the square root of numerous numbers. Filters 1, 2, 4, and 6 allow a time margin between measurements for use of the computed state estimates. They satisfy the definition of real-time application. Filters 3 and 5 do not. They are too slow to keep up with the tracker. Filter 6 does surprisingly well considering the number of matrix subroutine calls it must accomplish. Although slower than filters 1, 2, and 4, it provides a useful time margin between measurements.

Computational Precision

The comparison for computational precision is based on the normalized error (NE) between the final (100th iteration) state estimates from the 60-bit result (CDC-6600) and the 32-bit result (PDP-11/45). The normalized error test is defined as

$$NE = \sum_{i=1}^{9} \left| \frac{x_i^{60} - x_i^{32}}{x_i^{60}} \right| \qquad (37)$$

Table III

Computational Precision

Filter	1	2	3	4	5	6
N.E.	.00062	.00058	.00002	.00056	.00058	.00066

where x_i^{60} is the ith state estimate from the 60-bit result. The findings of this test are delineated in Table III. This test is designed to show round-off error trends due to finite computer word lengths. The lack of computer round-off error is indicated by a normalized error equal to zero.

Since filters 3, 4, and 5 are square root filters, their normalized errors should be less than filters 1, 2, and 6. The low normalized error for filter 3 contributes to the general lack of confidence in the filter. The normalized errors for filter 4 and 5 seem more reasonable when compared with the other filters. Filter 2 seems to be as good as filter 5 and much better than filters 1 and 6. Filter 6 again proves itself to be the worst case. Table III indicates that it suffers the most from computer round-off error. This condition may be aggravated by the observed development of a nonsymmetric covariance ($\bar{P}$) matrix in the filter program. This problem will require additional code to fix, resulting in more storage requirements and slower execution speed.

Conclusion

This study investigates the feasibility of adapting an

extended Kalman filter to the real-time satellite-to-satellite tracking problem. Six filters of linear propagation and nonlinear measurements are tested. They determine the state estimates (position, velocity, and acceleration) of the tracked satellite relative to the tracking satellite in a local inertial reference frame. The six filters employ different covariance update techniques and methods of processing the measurement vector. Filters one through five exploit the most current state estimates after each scalar measurement to improve the filters' performance. This is not possible with filter six. This feature is illustrated in the structure diagrams for the filters. Three filters qualify for real-time application, but filter four is felt to give the best overall performance with respect to computer program storage requirements, program execution speed, and computational precision.

The computer program storage requirements fell into two groups. The first group is made up of the first five filters. These filters are optimized for storage requirements. Matrices are reduced to a few constants. Small utility matrices and constants are used when possible. The second group consists of filter six. It is the worst case reference with its complete matrices and associated matrix subroutines. A saving of between 87 and 50 percent can be achieved by chosing one of the optimized filters over filter six.

Since the application of the extended Kalman filter is

to be real-time, program execution speed is a major consideration. The filter program must be able to keep up with the tracker and still allow time for the use of its state estimates. If the tracker supplies measurements every tenth of a second, then filters three and five are too slow for the real-time application. Because filters one, two, and four operate at about 0.05 second, any one of them will supply the state estimates in a timely manner. Filter six is eliminated because it does not allow nearly as much time as filters one, two, and four.

Computational precision is the final selection criteria. Besides being small and quick, the filter must maintain its accuracy. The normalized error test indicates trends in round-off error accumulations due to finite computer word lengths. The filter with the closest value to zero should have the best computational precision. Excluding filter three, filter four provides the most accurate state estimates.

Therefore, based on a comparison of computer program storage requirements, program execution speeds, and computational precision, filter four is the best filter to solve the real-time satellite-to-satellite tracking problem. Other contributions are the demonstrated savings in storage and execution time for the optimized filters and the documentation that remains for each filter studied.

Recommendations

There are a number of efforts that could continue the

work begun by this thesis. The first effort should be the tuning of the $\bar{R}$ and $\bar{Q}$ matrices for optimal filter performance. This will require considerable computer time to match the filter state estimates with the true system. Once the tuning of the filter is accomplished, a Monte Carlo analysis should be performed, since this is inherently a nonlinear estimation problem (Ref 5). This will test the filter's reaction to random noise sources that corrupt the measurements and to various trajectory scenarios.

Different rendezvous approaches should be investigated to study the effect on program execution speed and computational precision. If additional accuracy is required and the time margin is long enough, then the fast Matrix RSS propagation module could be changed to either a Householder's transformation or a modified Gram-Schmidt module (Ref 7:Ch 7, 16).

Although this thesis addresses the feasibility of adapting an extended Kalman filter to the real-time satellite-to-satellite tracking problem, it has another use. It can serve as an academic model for tracking any accelerating vehicle from another (Ref 4). In this respect, the long term benefits of this thesis might be the most useful.

Bibliography

1. Arpin, David A. State Noise Covariance Computation in the Kalman Filter. MS thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December 1977.

2. Bierman, Gerald J. Factorization Methods for Discrete Sequential Estimation. New York: Academic Press, Inc. 1977.

3. DEC-11-LFPUA-B-D. FORTRAN IV-PLUS User's Guide. Maynard, Massachusetts: Digital Equipment Corporation, 1975.

4. Gardner, William G. Application of an Extended Kalman Filter for Real Time Aircraft-to-Aircraft Tracking Utilizing Angle Information Derived from Triangulation Techniques. Unpublished thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, March, 1978.

5. Jackson, Kenneth L. A Generalized Monte Carlo Analysis Program for Kalman Filter Design with Application to an Aircraft-to-Satellite Tracking Filter. MS thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December 1977.

6. Maybeck, Peter S. Stochastic Estimation and Control Systems. Unpublished text. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, February 1975.

7. -----. Stochastic Models, Estimation and Control. Unpublished text. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, January, 1977. To appear: New York: Academic Press, Inc., December 1978.

8. PDP-11 04/05/10/35/40/45 Processor Handbook. Maynard, Massachusetts: Digital Equipment Corporation, 1975.

9. Pearson, John B., III and E. B. Stead. "Kalman Filter Applications in Airborne Radar Tracking," IEEE Transactions on Aerospace and Electronic Systems, Vol AES-10, No. 3, May 1974.

10. Reid, J. Gary. Consultations concerning extended Kalman filter equations. Wright-Patterson Air Force Base, Ohio. October 1977. (Conversation).

11. Yourdon, Edward and Larry L. Constantine. Structured Design. New York: Yourdon Inc., 1975.

Appendix A

Description of Typical Rendezvous

This is a representative time history of the 100 input measurements for an approaching satellite in an elliptical orbit (193 X 820 km) closing on a satellite in low circular orbit (193 X 193 km). Ranges (R) are in kilometers. Angles (AZ and EL) are in radians. Measurements begin upon initial acquisition and continue for ten seconds. Initially the x axis points toward the tracked satellite, thus making AZ and EL zero. For this rendezvous measurements of EL remain below the tracker's measurement threshold.

Seconds	R	AZ	EL
0.0	54.338	0.0000	0.0000
0.5	54.255	-0.0001	0.0000
1.0	54.172	-0.0001	0.0000
1.5	54.089	-0.0002	0.0000
2.0	54.006	-0.0002	0.0000
2.5	53.923	-0.0003	0.0000
3.0	53.840	-0.0004	0.0000
3.5	53.757	-0.0004	0.0000
4.0	53.674	-0.0005	0.0000
4.5	53.591	-0.0005	0.0000
5.0	53.508	-0.0006	0.0000
5.5	53.425	-0.0007	0.0000

Seconds	R	AZ	EL
6.0	53.341	-0.0007	0.0000
6.5	53.258	-0.0008	0.0000
7.0	53.175	-0.0008	0.0000
7.5	53.092	-0.0009	0.0000
8.0	53.009	-0.0009	0.0000
8.5	52.926	-0.0010	0.0000
9.0	52.842	-0.0011	0.0000
9.5	52.759	-0.0011	0.0000
10.0	52.676	-0.0012	0.0000

Appendix B

Extended Kalman Filter Matrices

The initial conditions for the untuned extended Kalman filters are

$$\bar{x}_0 = \begin{bmatrix} 54.338 \text{ km} \\ 0 \quad \text{km/sec} \\ 0 \quad \text{km/sec}^2 \\ 0 \quad \text{km} \\ 0 \quad \text{km/sec} \\ 0 \quad \text{km/sec}^2 \\ 0 \quad \text{km} \\ 0 \quad \text{km/sec} \\ 0 \quad \text{km/sec}^2 \end{bmatrix}$$

$$\bar{R} = \begin{bmatrix} .0001 \text{ km}^2 & 0 & 0 \\ 0 & .00001 \text{ rad}^2 & 0 \\ 0 & 0 & .00001 \text{ rad}^2 \end{bmatrix}$$

43

$$\tilde{P}_0 = \begin{bmatrix} 1\ \text{km}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .01(\text{km/sec})^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .001(\text{km/sec}^2)^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1\ \text{km}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .01(\text{km/sec})^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .001(\text{km/sec}^2)^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1\ \text{km}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .01(\text{km/sec})^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .001(\text{km/sec}^2)^2 \end{bmatrix}$$

$$\bar{Q} = \begin{bmatrix}
\frac{.0001}{km^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{.00001}{(km/sec)^2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{.000001}{(km/sec^2)^2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{.0001}{km^2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{.00001}{(km/sec)^2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{.000001}{(km/sec^2)^2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{.0001}{km^2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{.00001}{(km/sec)^2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{.000001}{(km/sec^2)^2}
\end{bmatrix}$$

Appendix C.1

Filter One: Conventional Kalman Filter Program

Filter one represents the simplest form of the extended Kalman filters that process the measurement vector one component at a time. It employs the covariance update technique of

$$\bar{P}(t_i^+) = \bar{P}(t_i^-) - Ka^T\bar{P}(t_i^-) \qquad (30)$$

Bierman implements this equation (Ref 2:21) in his linear Kalman data processing algorithm (Ref 2:29). His FORTRAN mechanization is modified in this program for the nonlinear system and is improved with respect to storage requirements and execution speed. Propagation is accomplished by utilizing Equations 11 and 13.

This program is written in American National Standards Institute (ANSI) FORTRAN, as described in X.3-1966 with the exception of the PROGRAM instruction. Program documentation consists of internal comment cards, Fig. 5, and Bierman's equations and FORTRAN mechanization (Ref 2:28-29).

```
      PROGRAM EXNC1(INPUT,OUTPUT,TAPE1=INPUT,TAPE2=OUTPUT)
      DIMENSION X(9),P(9,9),Z(3),A(9),G(9),V(9),ZP(3),R(3)
      REAL A,DELTA,DENOMC,G,P,PHI12,PHI13,PHI23,PHI33,QA,QV,QX,R,S,SIGMA
     C,TAU,TEMP,UA,V,VA,X,Z,ZP
      INTEGER I,IA,IB,IC,J,K,L,M
C
C     ******************************
C     *                            *
C     *     INITIATE MODULE        *
C     *                            *
C     * INPUT - NONE OUTPUT - X    *
C     *                     - P    *
C     *                     - Q    *
C     *                     - PHI  *
C     *                     - R    *
C     *                            *
C     ******************************
C
      DELTA = .1
      TAU = 10.
      TEMP = EXP( - DELTA / TAU )
      PHI12 = DELTA
      PHI13 = ( - TAU + DELTA + TAU * TEMP ) * TAU
      PHI23 = TAU * (1. - TEMP )
      PHI33 = TEMP
      QX = .0001
      QV = .00001
      QA = .000001
      DO 16 I=1,9
      DO 16 J=1,9
```

```
      P(I,J) = 0.0
16    CONTINUE
      P(1,1) = 1.
      P(2,2) = .01
      P(3,3) = .001
      P(4,4) = 1.
      P(5,5) = .01
      P(6,6) = .001
      P(7,7) = 1.
      P(8,8) = .01
      P(9,9) = .001
      X(1) = 54.338
      X(2) = 0.0
      X(3) = 0.0
      X(4) = 0.0
      X(5) = 0.0
      X(6) = 0.0
      X(7) = 0.0
      X(8) = 0.0
      X(9) = 0.0
      R(1) = .0001
      R(2) = .00001
      R(3) = .000001
C
      DO 900 M=1,100
C
C     ******************************
C     *                            *
C     *         READ MODULE        *
C     *                            *
```

```
C     * INPUT - NONE OUTPUT - Z   *
C     *                           *
C     *****************************
C
      READ (1,601) (Z(I),I=1,3)
601   FORMAT (3F13.4)
C
C     *****************************
C     *                           *
C     *    PROPAGATE MODULE       *
C     *                           *
C     * INPUT - X    OUTPUT - X   *
C     *       - P           - P   *
C     *       - Q                 *
C     *       - PHI               *
C     *                           *
C     *****************************
C
C
C     PROPAGATE X VECTOR
C
      DO 110 I=1,7,3
      J = I + 1
      K = I + 2
      X(I) = X(I) + PHI12 * X(J) + PHI13 * X(K)
      X(J) = X(J) + PHI23 * X(K)
      X(K) = PHI33 * X(K)
110   CONTINUE
C
C     PROPAGATE P MATRIX
```

```
C
      CO 115 I=1,7,3
      J = I + 1
      K = I + 2
      P(J,I) = P(I,J)
      P(K,I) = P(I,K)
      P(K,J) = P(J,K)
115   CONTINUE
C
      DO 120 IA=1,7,3
      IB = IA + 1
      IC = IA + 2
      DO 120 I = IA,7,3
      J = I + 1
      K = I + 2
      A(1) = P(IA,I)
      A(2) = P(IB,I)
      A(3) = P(IC,I)
      A(4) = P(IA,J)
      A(5) = P(IB,J)
      A(6) = P(IC,J)
      A(7) = P(IA,K)
      A(8) = P(IB,K)
      A(9) = P(IC,K)
      P(IA,I)=A(1)+PHI12*A(2)+PHI13*A(3)+PHI12*(A(4)+PHI12*A(5)+PHI13*
     CA(6))+PHI13*( A(7)+PHI12*A(8)+PHI13*A(9))
      P(IA,J)=A(4)+PHI12*A(5)+PHI13*A(6)+PHI23*(A(7)+PHI12*A(8)+PHI13*
     CA(9))
      P(IA,K)=PHI33*(A(7)+PHI12*A(8)+PHI13*A(9))
      P(IB,J)=A(5)+PHI23*(A(6)+A(8)+PHI23*A(9))
```

```
      P(IB,K)=PHI33*(A(8)+PHI23*A(9))
      P(IC,K)=PHI33*PHI33*A(9)
      IF ( IA .EQ. I ) GO TO 120
      P(IB,I)=A(2)+PHI23*A(3)+PHI12*(A(5)+PHI23*A(6))+PHI13*(A(8)+PHI23*
     CA(9))
      P(IC,I)=PHI33*(A(3)+PHI12*A(5)+PHI13*A(9))
      P(IC,J)=PHI33*(A(6)+PHI23*A(9))
120   CONTINUE
C
      DO 140 I=1,7,3
      J = I + 1
      K = I + 2
      P(I,I) = P(I,I) + QX
      P(J,J) = P(J,J) + QV
      P(K,K) = P(K,K) + QA
140   CONTINUE
C
C     ******************************
C     *                            *
C     *       UPDATE MODULE        *
C     *                            *
C     * INPUT - Z    OUTPUT - X    *
C     *       - X           - P    *
C     *       - P                  *
C     *       - R                  *
C     *                            *
C     ******************************
C
      DO 290 I=2,9
      A(I) = 0.0
```

```
290   CONTINUE
C
      DO 50 L=1,3
C
C     COMPUTE ZP AND H
C
      IF ( L .NE. 1 ) GO TO 210
      TEMP   = SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      ZP(1) = TEMP
      TEMP = 1. / TEMP
      A(1) = X(1) * TEMP
      A(4) = X(4) * TEMP
      A(7) = X(7) * TEMP
      GO TO 230
210   IF ( L .NE. 2 ) GO TO 220
      UA = X(4) / X(1)
      TEMP = 1. / ((1. + UA * UA) * X(1))
      ZP(2) = ATAN(UA)
      A(1) = - UA * TEMP
      A(4) = TEMP
      A(7) = 0.0
      GO TO 230
220   TEMP = 1. / SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      VA = X(7) * TEMP
      DENOMC = SQRT(1. - VA * VA)
      ZP(3) = ASIN(VA)
      A(1) = - VA * X(1) * TEMP * TEMP / DENOMC
      A(4) = - VA * X(4) * TEMP * TEMP / DENOMC
      A(7) = DENOMC * TEMP
C
```

```
C       UPDATE P,X
C
230     SIGMA = R(L)
        DELTA = Z(L) - ZP(L)
        V(1) = P(1,1) * A(1) + P(1,4) * A(4) + P(1,7) * A(7)
        V(2) = P(1,2) * A(1) + P(2,4) * A(4) + P(2,7) * A(7)
        V(3) = P(1,3) * A(1) + P(3,4) * A(4) + P(3,7) * A(7)
        V(4) = P(1,4) * A(1) + P(4,4) * A(4) + P(4,7) * A(7)
        V(5) = P(1,5) * A(1) + P(4,5) * A(4) + P(5,7) * A(7)
        V(6) = P(1,6) * A(1) + P(4,6) * A(4) + P(6,7) * A(7)
        V(7) = P(1,7) * A(1) + P(4,7) * A(4) + P(7,7) * A(7)
        V(8) = P(1,8) * A(1) + P(4,8) * A(4) + P(7,8) * A(7)
        V(9) = P(1,9) * A(1) + P(4,9) * A(4) + P(7,9) * A(7)
        DO 10 I=1,7,3
        SIGMA = SIGMA + A(I) * V(I)
10      CONTINUE
C
        SIGMA = 1. / SIGMA
        DO 20 I=1,9
        G(I) = V(I) * SIGMA
        X(I) = X(I) + G(I) * DELTA
        DO 20 J=I,9
        P(I,J) = P(I,J) - G(I) * V(J)
20      CONTINUE
50      CONTINUE
C
C       ******************************
C       *                            *
C       *        WRITE MODULE        *
C       *                            *
```

```
C      * INPUT - X     OUTPUT - NONE*
C      *                             *
C      *******************************
C
       WRITE (2,602) (X(I),I=1,9)
602    FORMAT ( F20.15 )
900    CONTINUE
       STOP
       END
```

Appendix C.2

Filter Two: Stabilized Kalman Filter Program

Filter two is a modification of filter one. It also processes the measurement vector one component at a time, but employs the covariance update technique of

$$\bar{P}(t_i^+) = (I-Ka^T)\bar{P}(t_i^-)(I-Ka^T)^T + KrK^T \qquad (31)$$

for better accuracy (Ref 2:20). Bierman implements this equation in his linear Kalman data processing algorithm (Ref 2:29). His FORTRAN mechanization is modified in this program for the nonlinear system and is improved with respect to storage requirements and execution speed. Propagation is accomplished by utilizing Equations 11 and 13.

This program is written in American National Standards Institute (ANSI) FORTRAN, as described in X.3-1966 with the exception of the PROGRAM instruction. Program documentation consists of internal comment cards, Fig. 6, and Bierman's equations and FORTRAN mechanization (Ref 2:28-29).

```
      PROGRAM EXN02(INPUT,OUTPUT,TAPE1=INPUT,TAPE2=OUTPUT)
      DIMENSION X(9),P(9,9),Z(3),A(9),G(9),V(9),ZP(3),R(3)
      REAL A,DELTA,DENOMC,G,P,PHI12,PHI13,PHI23,PHI33,QA,QV,QX,R,S,SIGMA
     C,TAU,TEMP,UA,V,VA,X,Z,ZP,S
      INTEGER I,IA,IB,IC,J,K,L,M
C
C     ********************************
C     *                              *
C     *     INITIATE MODULE          *
C     *                              *
C     * INPUT - NONE OUTPUT - X      *
C     *                     - P      *
C     *                     - Q      *
C     *                     - PHI    *
C     *                     - R      *
C     *                              *
C     ********************************
C
      DELTA = .1
      TAU = 10.
      TEMP = EXP( - DELTA / TAU )
      PHI12 = DELTA
      PHI13 = ( - TAU + DELTA + TAU * TEMP ) * TAU
      PHI23 = TAU * (1. - TEMP )
      PHI33 = TEMP
      QX = .0001
      QV = .00001
      QA = .000001
      DO 16 I=1,9
      DO 16 J=1,9
```

```
      P(I,J) = 0.0
16    CONTINUE
      P(1,1) = 1.
      P(2,2) = .01
      P(3,3) = .001
      P(4,4) = 1.
      P(5,5) = .01
      P(6,6) = .001
      P(7,7) = 1.
      P(8,8) = .01
      P(9,9) = .001
      X(1) = 54.338
      X(2) = 0.0
      X(3) = 0.0
      X(4) = 0.0
      X(5) = 0.0
      X(6) = 0.0
      X(7) = 0.0
      X(8) = 0.0
      X(9) = 0.0
      R(1) = .0001
      R(2) = .00001
      R(3) = .00001
C
      DO 900 M=1,100
C
C     ********************************
C     *                              *
C     *         READ MODULE          *
C     *                              *
```

```
C      * INPUT - NONE OUTPUT - Z    *
C      *                            *
C      ******************************
C
       READ (1,601) (Z(I),I=1,3)
601    FORMAT (3F13.4)
C
C      ******************************
C      *                            *
C      *    PROPAGATE MODULE        *
C      *                            *
C      * INPUT - X    OUTPUT - X    *
C      *       - P             - P  *
C      *       - Q                  *
C      *       - PHI                *
C      *                            *
C      ******************************
C
C
C      PROPAGATE X VECTOR
C
       DO 110 I=1,7,3
       J = I + 1
       K = I + 2
       X(I) = X(I) + PHI12 * X(J) + PHI13 * X(K)
       X(J) = X(J) + PHI23 * X(K)
       X(K) = PHI33 * X(K)
110    CONTINUE
C
C      PROPAGATE P MATRIX
```

```
C
      DO 115 I=1,7,3
      J = I + 1
      K = I + 2
      P(J,I) = P(I,J)
      P(K,I) = P(I,K)
      P(K,J) = P(J,K)
115   CONTINUE
C
      DO 120 IA=1,7,3
      IB = IA + 1
      IC = IA + 2
      DO 120 I = IA,7,3
      J = I + 1
      K = I + 2
      A(1) = P(IA,I)
      A(2) = P(IB,I)
      A(3) = P(IC,I)
      A(4) = P(IA,J)
      A(5) = P(IB,J)
      A(6) = P(IC,J)
      A(7) = P(IA,K)
      A(8) = P(IB,K)
      A(9) = P(IC,K)
      P(IA,I)=A(1)+PHI12*A(2)+PHI13*A(3)+PHI12*(A(4)+PHI12*A(5)+PHI13*
     CA(6))+PHI13*( A(7)+PHI12*A(8)+PHI13*A(9))
      P(IA,J)=A(4)+PHI12*A(5)+PHI13*A(6)+PHI23*(A(7)+PHI12*A(8)+PHI13*
     CA(9))
      P(IA,K)=PHI33*(A(7)+PHI12*A(8)+PHI13*A(9))
      P(IB,J)=A(5)+PHI23*(A(6)+A(3)+PHI23*A(9))
```

```
      P(IB,K)=PHI33*(A(8)+PHI23*A(9))
      P(IC,K)=PHI33*PHI33*A(9)
      IF ( IA .EQ. I ) GO TO 120
      P(IB,I)=A(2)+PHI23*A(3)+PHI12*(A(5)+PHI23*A(6))+PHI13*(A(8)+PHI23*
     CA(9))
      P(IC,I)=PHI33*(A(3)+PHI12*A(5)+PHI13*A(9))
      P(IC,J)=PHI33*(A(6)+PHI23*A(9))
120   CONTINUE
C
      DO 140 I=1,7,3
      J = I + 1
      K = I + 2
      P(I,I) = P(I,I) + QX
      P(J,J) = P(J,J) + QV
      P(K,K) = P(K,K) + QA
140   CONTINUE
C
C     ********************************
C     *                              *
C     *        UPDATE MODULE         *
C     *                              *
C     * INPUT - Z     OUTPUT - X     *
C     *         - X              - P *
C     *         - P                  *
C     *         - R                  *
C     *                              *
C     ********************************
C
      DO 290 I=2,9
      A(I) = 0.0
```

```
290   CONTINUE
C
      DO 50 L=1,3
C
C     COMPUTE ZP AND H
C
      IF ( L .NE. 1 ) GO TO 210
      TEMP   = SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      ZP(1) = TEMP
      TEMP = 1. / TEMP
      A(1) = X(1) * TEMP
      A(4) = X(4) * TEMP
      A(7) = X(7) * TEMP
      GO TO 230
210   IF ( L .NE. 2 ) GO TO 220
      UA = X(4) / X(1)
      TEMP = 1. / ((1. + UA * UA) * X(1))
      ZP(2) = ATAN(UA)
      A(1) = - UA * TEMP
      A(4) = TEMP
      A(7) = 0.0
      GO TO 230
220   TEMP = 1. / SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      VA = X(7) * TEMP
      DENOMC = SQRT(1. - VA * VA)
      ZP(3) = ASIN(VA)
      A(1) = - VA * X(1) * TEMP * TEMP / DENOMC
      A(4) = - VA * X(4) * TEMP * TEMP / DENOMC
      A(7) = DENOMC * TEMP
C
```

```
C       UPDATE P,X
C
230     SIGMA = R(L)
        DELTA = Z(L) - ZP(L)
        V(1) = P(1,1) * A(1) + P(1,4) * A(4) + P(1,7) * A(7)
        V(2) = P(1,2) * A(1) + P(2,4) * A(4) + P(2,7) * A(7)
        V(3) = P(1,3) * A(1) + P(3,4) * A(4) + P(3,7) * A(7)
        V(4) = P(1,4) * A(1) + P(4,4) * A(4) + P(4,7) * A(7)
        V(5) = P(1,5) * A(1) + P(4,5) * A(4) + P(5,7) * A(7)
        V(6) = P(1,6) * A(1) + P(4,6) * A(4) + P(6,7) * A(7)
        V(7) = P(1,7) * A(1) + P(4,7) * A(4) + P(7,7) * A(7)
        V(8) = P(1,8) * A(1) + P(4,8) * A(4) + P(7,8) * A(7)
        V(9) = P(1,9) * A(1) + P(4,9) * A(4) + P(7,9) * A(7)
        DO 10 I=1,7,3
        SIGMA = SIGMA + A(I) * V(I)
10      CONTINUE
C
        SIGMA = 1. / SIGMA
        DO 20 I=1,9
        G(I) = V(I) * SIGMA
        X(I) = X(I) + G(I) * DELTA
        DO 20 J=I,9
        P(I,J) = P(I,J) - G(I) * V(J)
20      CONTINUE
C
        V(1) = P(1,1) * A(1) + P(1,4) * A(4) + P(1,7) * A(7)
        V(2) = P(1,2) * A(1) + P(2,4) * A(4) + P(2,7) * A(7)
        V(3) = P(1,3) * A(1) + P(3,4) * A(4) + P(3,7) * A(7)
        V(4) = P(1,4) * A(1) + P(4,4) * A(4) + P(4,7) * A(7)
        V(5) = P(1,5) * A(1) + P(4,5) * A(4) + P(5,7) * A(7)
```

```
      V(6) = P(1,6) * A(1) + P(4,6) * A(4) + P(6,7) * A(7)
      V(7) = P(1,7) * A(1) + P(4,7) * A(4) + P(7,7) * A(7)
      V(8) = P(1,8) * A(1) + P(4,8) * A(4) + P(7,8) * A(7)
      V(9) = P(1,9) * A(1) + P(4,9) * A(4) + P(7,9) * A(7)
      DO 40 J=1,9
      DO 40 I=1,J
      S = .5 * (P(I,J) - V(I) * G(J) + P(I,J) - V(J) * G(I))
      P(I,J) = S + G(I) * R(L) * G(J)
40    CONTINUE
50    CONTINUE
C
C     ********************************
C     *                              *
C     *        WRITE MODULE          *
C     *                              *
C     * INPUT - X    OUTPUT - NONE   *
C     *                              *
C     ********************************
C
      WRITE (2,602) (X(I),I=1,9)
602   FORMAT ( F20.15 )
900   CONTINUE
      STOP
      END
```

Appendix C.3

Filter Three: Modified Potter Filter Program

Filter three is a modification of the Potter algorithm. It processes the measurement vector one component at a time and employs the covariance update technique of

$$\bar{P}(t_i^+) = \bar{S}(t_i^+)\bar{S}(t_i^+)^T \qquad (32)$$

where $\bar{S}(t_i^+)$ is calculated by Bierman's FORTRAN mechanization of Potter's data processing method for the linear system (Ref 2:31). His mechanization is modified in this program for the nonlinear system and is improved with respect to storage requirements and execution speed. Propagation is accomplished by the Matrix RSS method utilizing Equations 11 and 13 (Ref 7:Ch 7, 16.)

This program is written in American National Standards Institute (ANSI) FORTRAN, as described in X.3-1966 with the exception of the PROGRAM instruction. Program documentation consists of internal comment cards, Fig. 7, Bierman's equations and FORTRAN mechanization (Ref 2:30-31).

```
      PROGRAM EXNO3(INPUT,OUTPUT,TAPE1=INPUT,TAPE2=OUTPUT)
      DIMENSION X(9),P(9,9),S(9,9),Z(3),A(9),V(9),ZP(3),R(3)
      REAL A,ALPHA,BETA,DELTA,DENOMC,GAMMA,P,PHI12,PHI13,PHI23,PHI33,
     COA,OV,OX,P,S,SIGMA,TAU,TEMP,JA,V,VA,X,Z,ZP
      INTEGER I,IA,IB,IC,I1,I2,J,K,L,M
C
C     ******************************
C     *                            *
C     *     INITIATE MODULE        *
C     *                            *
C     * INPUT - NONE OUTPUT - X    *
C     *                     - P    *
C     *                     - Q    *
C     *                     - PHI  *
C     *                     - R    *
C     *                            *
C     ******************************
C
      DELTA = .1
      TAU = 10.
      TEMP = EXP( - DELTA / TAU )
      PHI12 = DELTA
      PHI13 = ( - TAU + DELTA + TAU * TEMP ) * TAU
      PHI23 = TAU * (1. - TEMP )
      PHI33 = TEMP
      OX = .0001
      OV = .00001
      OA = .000001
      DO 16 I=1,9
      DO 16 J=1,9
```

```
      P(I,J) = 0.0
16    CONTINUE
      P(1,1) = 1.
      P(2,2) = .01
      P(3,3) = .001
      P(4,4) = 1.
      P(5,5) = .01
      P(6,5) = .001
      P(7,7) = 1.
      P(8,8) = .01
      P(9,9) = .001
      X(1) = 54.338
      X(2) = 0.0
      X(3) = 0.0
      X(4) = 0.0
      X(5) = 0.0
      X(6) = 0.0
      X(7) = 0.0
      X(8) = 0.0
      X(9) = 0.0
      R(1) = .0001
      R(2) = .00001
      R(3) = .000001
C
      DO 900 M=1,100
C
C     ********************************
C     *                              *
C     *         READ MODULE          *
C     *                              *
```

```
C     * INPUT - NONE OUTPUT - Z   *
C     *                           *
C     *****************************
C
      READ (1,601) (Z(I),I=1,3)
601   FORMAT (3F13.4)
C
C     *****************************
C     *                           *
C     *    PROPAGATE MODULE       *
C     *                           *
C     * INPUT - X    OUTPUT - X   *
C     *       - P           - P   *
C     *       - Q                 *
C     *       - PHI               *
C     *                           *
C     *****************************
C
C
C     PROPAGATE X VECTOR
C
      DO 110 I=1,7,3
      J = I + 1
      K = I + 2
      X(I) = X(I) + PHI12 * X(J) + PHI13 * X(K)
      X(J) = X(J) + PHI23 * X(K)
      X(K) = PHI33 * X(K)
110   CONTINUE
C
C     PROPAGATE P MATRIX
```

```
C
      DO 115 I=1,7,3
      J = I + 1
      K = I + 2
      P(J,I) = P(I,J)
      P(K,I) = P(I,K)
      P(K,J) = P(J,K)
115   CONTINUE
C
      DO 120 IA=1,7,3
      IB = IA + 1
      IC = IA + 2
      DO 120 I = IA,7,3
      J = I + 1
      K = I + 2
      A(1) = P(IA,I)
      A(2) = P(IB,I)
      A(3) = P(IC,I)
      A(4) = P(IA,J)
      A(5) = P(IB,J)
      A(6) = P(IC,J)
      A(7) = P(IA,K)
      A(8) = P(IB,K)
      A(9) = P(IC,K)
      P(IA,I)=A(1)+PHI12*A(2)+PHI13*A(3)+PHI12*(A(4)+PHI12*A(5)+PHI13*
     CA(6))+PHI13*( A(7)+PHI12*A(8)+PHI13*A(9))
      P(IA,J)=A(4)+PHI12*A(5)+PHI13*A(6)+PHI23*(A(7)+PHI12*A(8)+PHI13*
     CA(9))
      P(IA,K)=PHI33*(A(7)+PHI12*A(8)+PHI13*A(9))
      P(IB,J)=A(5)+PHI23*(A(6)+A(8)+PHI23*A(9))
```

```
      P(IB,K)=PHI33*(A(8)+PHI23*A(9))
      P(IC,K)=PHI33*PHI33*A(9)
      IF ( IA .EQ. I ) GO TO 120
      P(IB,I)=A(2)+PHI23*A(3)+PHI12*(A(5)+PHI23*A(6))+PHI13*(A(8)+PHI23*
     CA(9))
      P(IC,I)=PHI33*(A(3)+PHI12*A(6)+PHI13*A(9))
      P(IC,J)=PHI33*(A(6)+PHI23*A(9))
120   CONTINUE
C
      DO 140 I=1,7,3
      J = I + 1
      K = I + 2
      P(I,I) = P(I,I) + QX
      P(J,J) = P(J,J) + QV
      P(K,K) = P(K,K) + QA
140   CONTINUE
C
      DO 130 I=1,9
      DO 130 J=I,9
      P(J,I) = P(I,J)
130   CONTINUE
C
C     ******************************
C     *                            *
C     *       UPDATE MODULE        *
C     *                            *
C     * INPUT - Z    OUTPUT - X    *
C     *       - X           - P    *
C     *       - P                  *
C     *       - R                  *
```

```
C     *                              *
C     ********************************
C
C
C     LOWER TRIANGULAR CHOLESKY FACTORIZATION
C
      DO 17 I=1,9
      DO 17 J=1,9
      S(I,J) = 0.0
17    CONTINUE
C
      DO 25 J=1,8
      S(J,J) = SQRT (P(J,J))
      ALPHA = 1. / S(J,J)
      I1 = J + 1
      DO 25 I2 = I1,9
      K = 9 + I1 - I2
      S(K,J) = ALPHA * P(K,J)
      BETA = S(K,J)
      DO 25 I=K,9
      P(I,K) = P(I,K) - S(I,J) * BETA
25    CONTINUE
      S(9,9) = SQRT (P(9,9))
      DO 290 I=2,9
      A(I) = 0.0
290   CONTINUE
C
      DO 50 L=1,3
C
C     COMPUTE ZP AND H
```

```
C
      IF ( L .NE. 1 ) GO TO 210
      TEMP   = SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      ZP(1) = TEMP
      TEMP = 1. / TEMP
      A(1) = X(1) * TEMP
      A(4) = X(4) * TEMP
      A(7) = X(7) * TEMP
      GO TO 230
210   IF ( L .NE. 2 ) GO TO 220
      UA = X(4) / X(1)
      TEMP = 1. / ((1. + UA * UA) * X(1))
      ZP(2) = ATAN(UA)
      A(1) = - UA * TEMP
      A(4) = TEMP
      A(7) = 0.0
      GO TO 230
220   TEMP = 1. / SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      VA = X(7) * TEMP
      DENOMC = SQRT(1. - VA * VA)
      ZP(3) = ASIN(VA)
      A(1) = - VA * X(1) * TEMP * TEMP / DENOMC
      A(4) = - VA * X(4) * TEMP * TEMP / DENOMC
      A(7) = DENOMC * TEMP
C
C
230   SIGMA = R(L)
      DELTA = Z(L) - ZP(L)
      DO 10 I=1,9
      V(I) = 0.
```

```
      DO 5 J=1,7,3
      V(I) = V(I) + S(J,I) * A(J)
5     CONTINUE
      SIGMA = SIGMA + V(I) * V(I)
10    CONTINUE
C
      SIGMA = 1. / SIGMA
      DELTA = DELTA * SIGMA
C     UPDATE S,X
      GAMMA = SIGMA / ( 1. + SQRT (SIGMA))
      DO 20 I=1,9
      ALPHA = 0.
      DO 15 J=1,9
      ALPHA = ALPHA + S(I,J) * V(J)
15    CONTINUE
      X(I) = X(I) + ALPHA * DELTA
      ALPHA = ALPHA * GAMMA
      DO 20 J=1,9
      S(I,J) = S(I,J) - ALPHA * V(J)
20    CONTINUE
50    CONTINUE
C
C     CONVERT P
C
      DO 300 I=1,9
      DO 300 J=1,I
      TEMP = 0.0
      DO 350 K=1,9
      TEMP = TEMP + S(I,K) * S(J,K)
350   CONTINUE
```

```
      P(J,I) = TEMP
300   CONTINUE
C
C     ********************************
C     *                              *
C     *        WRITE MODULE          *
C     *                              *
C     * INPUT - X    OUTPUT - NONE   *
C     *                              *
C     ********************************
C
      WRITE (2,602) (X(I),I=1,9)
602   FORMAT ( F20.15 )
900   CONTINUE
      STOP
      END
```

Appendix C.4

Filter Four: Modified U-D Filter Program

Filter four is a modification of the U-D algorithm. It processes the measurement vector one component at a time and employs the covariance update technique of

$$\bar{P}(t_i^+) = \bar{U}(t_i^+)\bar{D}(t_i^+)\bar{U}(t_i^+)^T \qquad (33)$$

where $\bar{U}(t_i^+)$ and $\bar{D}(t_i^+)$ are calculated by Bierman's FORTRAN mechanization in the $\bar{U}$-$\bar{D}(t_i^+)$ form for the linear system (Ref 2:101). His mechanization is modified in this program for the nonlinear system and is improved with respect to storage requirements and execution speed. Propagation is accomplished by the Matrix RSS method utilizing Equations 11 and 13 (Ref 7:Ch 7, 16).

This program is written in American National Standards Institute (ANSI) FORTRAN, as described in X.3-1966 with the exception of the PROGRAM instruction. Program documentation consists of internal comment cards, Fig. 8, Bierman's equations (Ref 2:77-80) and FORTRAN mechanization (Ref 2:101).

```
      PROGRAM EXN04(INPUT,OUTPUT,TAPE1=INPUT,TAPE2=OUTPUT)
      DIMENSION X(9),P(9,9),U(9,9),Z(3),A(9),B(9),ZP(3),R(3),D(9)
      REAL A,ALPHA,B,BETA,D,DELTA,DENOMC,GAMMA,LAMBDA,P,PHI12,PHI13,
     C PHI23,PHI33,QA,QV,QX,R,TAU,TEMP,U,UA,VA,X,Z,ZP
      INTEGER I,IA,IB,IC,I1,I2,J,K,L,M
C
C     ********************************
C     *                              *
C     *     INITIATE MODULE          *
C     *                              *
C     * INPUT - NONE OUTPUT - X      *
C     *                     - P      *
C     *                     - Q      *
C     *                     - PHI    *
C     *                     - R      *
C     *                              *
C     ********************************
C
      DELTA = .1
      TAU = 10.
      TEMP = EXP( - DELTA / TAU )
      PHI12 = DELTA
      PHI13 = ( - TAU + DELTA + TAU * TEMP ) * TAU
      PHI23 = TAU * (1. - TEMP )
      PHI33 = TEMP
      QX = .0001
      QV = .00001
      QA = .000001
      DO 16 I=1,9
      DO 16 J=1,9
```

```
      P(I,J) = 0.0
16    CONTINUE
      P(1,1) = 1.
      P(2,2) = .01
      P(3,3) = .001
      P(4,4) = 1.
      P(5,5) = .01
      P(6,6) = .001
      P(7,7) = 1.
      P(8,8) = .01
      P(9,9) = .001
      X(1) = 54.338
      X(2) = 0.0
      X(3) = 0.0
      X(4) = 0.0
      X(5) = 0.0
      X(6) = 0.0
      X(7) = 0.0
      X(8) = 0.0
      X(9) = 0.0
      R(1) = .0001
      R(2) = .00001
      R(3) = .00001
C
      DO 900 M=1,100
C
C     ********************************
C     *                              *
C     *         READ MODULE          *
C     *                              *
```

```
C      * INPUT - NONE OUTPUT - Z    *
C      *                            *
C      ******************************
C
       READ (1,601) (Z(I),I=1,3)
601    FORMAT (3F13.4)
C
C      ******************************
C      *                            *
C      *    PROPAGATE MODULE        *
C      *                            *
C      * INPUT - X     OUTPUT - X   *
C      *       - P              - P *
C      *       - Q                  *
C      *       - PHI                *
C      *                            *
C      ******************************
C
C
C      PROPAGATE X VECTOR
C
       DO 110 I=1,7,3
       J = I + 1
       K = I + 2
       X(I) = X(I) + PHI12 * X(J) + PHI13 * X(K)
       X(J) = X(J) + PHI23 * X(K)
       X(K) = PHI33 * X(K)
110    CONTINUE
C
C      PROPAGATE P MATRIX
```

```
C
      DO 115 I=1,7,3
      J = I + 1
      K = I + 2
      P(J,I) = P(I,J)
      P(K,I) = P(I,K)
      P(K,J) = P(J,K)
115   CONTINUE
C
      DO 120 IA=1,7,3
      IB = IA + 1
      IC = IA + 2
      DO 120 I = IA,7,3
      J = I + 1
      K = I + 2
      A(1) = P(IA,I)
      A(2) = P(IB,I)
      A(3) = P(IC,I)
      A(4) = P(IA,J)
      A(5) = P(IB,J)
      A(6) = P(IC,J)
      A(7) = P(IA,K)
      A(8) = P(IB,K)
      A(9) = P(IC,K)
      P(IA,I)=A(1)+PHI12*A(2)+PHI13*A(3)+PHI12*(A(4)+PHI12*A(5)+PHI13*
     CA(6))+PHI13*( A(7)+PHI12*A(8)+PHI13*A(9))
      P(IA,J)=A(4)+PHI12*A(5)+PHI13*A(6)+PHI23*(A(7)+PHI12*A(8)+PHI13*
     CA(9))
      P(IA,K)=PHI33*(A(7)+PHI12*A(8)+PHI13*A(9))
      P(IB,J)=A(5)+PHI23*(A(6)+A(8)+PHI23*A(9))
```

```
      P(IB,K)=PHI33*(A(8)+PHI23*A(9))
      P(IC,K)=PHI33*PHI33*A(9)
      IF ( IA .EQ. I ) GO TO 120
      P(IB,I)=A(2)+PHI23*A(3)+PHI12*(A(5)+PHI23*A(6))+PHI13*(A(8)+PHI23*
     CA(9))
      P(IC,I)=PHI33*(A(3)+PHI12*A(6)+PHI13*A(9))
      P(IC,J)=PHI33*(A(6)+PHI23*A(9))
120   CONTINUE
C
      DO 140 I=1,7,3
      J = I + 1
      K = I + 2
      P(I,I) = P(I,I) + QX
      P(J,J) = P(J,J) + QV
      P(K,K) = P(K,K) + QA
140   CONTINUE
C
C     ********************************
C     *                              *
C     *       UPDATE MODULE          *
C     *                              *
C     * INPUT - Z    OUTPUT - X      *
C     *       - X           - P      *
C     *       - P                    *
C     *       - R                    *
C     *                              *
C     ********************************
C
C
C     UPPER TRIANGULAR UDUT FACTORIZATION
```

```
C
      DO 17 I=1,9
      DO 17 J=I,9
      U(I,J) = 0.0
17    CONTINUE
C
      DO 25 I2 =2,9
      J = 11 - I2
      D(J) = P(J,J)
      ALPHA = 1. / D(J)
      L = J - 1
      DO 25 K=1,L
      BETA = P(K,J)
      U(K,J) = ALPHA * BETA
      DO 25 I =1,K
      P(I,K) = P(I,K) - BETA * U(I,J)
25    CONTINUE
      D(1) = P(1,1)
      DO 35 I=1,9
      U(I,I) = D(I)
35    CONTINUE
C
      DO 50 L=1,3
C
      DO 290 I=2,9
      A(I) = 0.0
290   CONTINUE
C
C     COMPUTE ZP AND H
C
```

```
      IF ( L .NE. 1 ) GO TO 210
      TEMP   = SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      ZP(1) = TEMP
      TEMP = 1. / TEMP
      A(1) = X(1) * TEMP
      A(4) = X(4) * TEMP
      A(7) = X(7) * TEMP
      GO TO 230
210   IF ( L .NE. 2 ) GO TO 220
      UA = X(4) / X(1)
      TEMP = 1. / ((1. + UA * UA) * X(1))
      ZP(2) = ATAN(UA)
      A(1) = - UA * TEMP
      A(4) = TEMP
      GO TO 230
220   TEMP = 1. / SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      VA = X(7) * TEMP
      DENOMC = SQRT(1. - VA * VA)
      ZP(3) = ASIN(VA)
      A(1) = - VA * X(1) * TEMP * TEMP / DENOMC
      A(4) = - VA * X(4) * TEMP * TEMP / DENOMC
      A(7) = DENOMC * TEMP
C
C     UPDATE U-D,X
C
230   Z(L) = Z(L) - ZP(L)
      DO 10 I2 =2,9
      J = 11 - I2
      I = J - 1
      DO  5 K=1,I,3
```

```
      A(J) = A(J) + U(K,J) * A(K)
5     CONTINUE
      B(J) = U(J,J) * A(J)
10    CONTINUE
      B(1) = U(1,1) * A(1)
C
      ALPHA = R(L) + B(1) * A(1)
      GAMMA = 1. / ALPHA
      U(1,1) = R(L) * GAMMA * U(1,1)
      DO 20 J=2,9
      BETA = ALPHA
      ALPHA = ALPHA + B(J) * A(J)
      LAMBDA = -A(J) * GAMMA
      GAMMA = 1. / ALPHA
      U(J,J) = BETA * GAMMA * U(J,J)
C
      I1 = J - 1
      DO 20 I=1,I1
      BETA = U(I,J)
      U(I,J) = BETA + B(I) * LAMBDA
      B(I) = B(I) + B(J) * BETA
20    CONTINUE
      Z(L) = Z(L) * GAMMA
      DO 30 J=1,9
      X(J) = X(J) + B(J) * Z(L)
30    CONTINUE
50    CONTINUE
C
C     CONVERT P
C
```

```
      DO 310 I=1,9
      D(I) = U(I,I)
      U(I,I) = 1.0
310   CONTINUE
      DO 300 I=1,9
      DO 300 J=1,I
      TEMP = 0.0
      DO 350 K=I,9
      TEMP = TEMP + U(I,K) * D(K) * U(J,K)
350   CONTINUE
      P(J,I) = TEMP
300   CONTINUE
C
C     ******************************
C     *                            *
C     *        WRITE MODULE        *
C     *                            *
C     * INPUT - X    OUTPUT - NONE *
C     *                            *
C     ******************************
C
      WRITE (2,602) (X(I),I=1,9)
602   FORMAT ( F20.15 )
900   CONTINUE
      STOP
      END
```

Appendix C.5

Filter Five: Modified Upper Triangular Filter Program

Filter five is a modification of the upper triangular algorithm. It processes the measurement vector one component at a time and employs the covariance update technique of

$$\bar{P}(t_i^+) = \bar{U}(t_i^+)\bar{U}(t_i^+)^T \qquad (34)$$

where $\bar{U}(t_i^+)$ is calculated by Bierman's FORTRAN mechanization for the linear system (Ref 2:102). His mechanization is modified in this program for the nonlinear system and is improved with respect to storage requirements and execution speed. Propagation is accomplished by the Matrix RSS method utilizing Equations 11 and 13 (Ref 7:Ch 7, 16).

This program is written in American National Standards Institute (ANSI) FORTRAN, as described in X.3-1966 with the exception of the PROGRAM instruction. Program documentation consists of internal comment cards, Fig. 9, and Bierman's equations (Ref 2:81) and FORTRAN mechanization (Ref 2:102).

```
      PROGRAM EXN05(INPUT,OUTPUT,TAPE1=INPUT,TAPE2=OUTPUT)
      DIMENSION X(9),P(9,9),S(9,9),Z(3),A(9),B(9),ZP(3),R(3)
      REAL A,ALPHA,B,BETA,DELTA,DENOMC,GAMMA,P,PHI12,PHI13,PHI23,PHI33,
     C QA,QV,QX,R,S,SIGMA,T,TAU,TEMP,UA,VA,X,Z,ZP
      INTEGER I,IA,IB,IC,I1,I2,J,K,L,M
C
C     ******************************
C     *                            *
C     *     INITIATE MODULE        *
C     *                            *
C     * INPUT - NONE OUTPUT - X    *
C     *                     - P    *
C     *                     - Q    *
C     *                     - PHI  *
C     *                     - R    *
C     *                            *
C     ******************************
C
      DELTA = .1
      TAU = 10.
      TEMP = EXP( - DELTA / TAU )
      PHI12 = DELTA
      PHI13 = ( - TAU + DELTA + TAU * TEMP ) * TAU
      PHI23 = TAU * (1. - TEMP )
      PHI33 = TEMP
      QX = .0001
      QV = .00001
      QA = .000001
      DO 16 I=1,9
      DO 16 J=1,9
```

```
      P(I,J) = 0.0
16    CONTINUE
      P(1,1) = 1.
      P(2,2) = .01
      P(3,3) = .001
      P(4,4) = 1.
      P(5,5) = .01
      P(6,6) = .001
      P(7,7) = 1.
      P(8,8) = .01
      P(9,9) = .001
      X(1) = 54.338
      X(2) = 0.0
      X(3) = 0.0
      X(4) = 0.0
      X(5) = 0.0
      X(6) = 0.0
      X(7) = 0.0
      X(8) = 0.0
      X(9) = 0.0
      R(1) = .0001
      R(2) = .00001
      R(3) = .000001
C
      DO 900 M=1,100
C
C     ********************************
C     *                              *
C     *         READ MODULE          *
C     *                              *
```

AD-A055 115 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 17/9
INVESTIGATION OF REAL-TIME SATELLITE-TO-SATELLITE TRACKING USIN--ETC(U)
MAR 78 H J LAUGHLIN
UNCLASSIFIED AFIT/GCS/EE/78-3 NL

2 OF 2
AD
A055 115

END
DATE
FILMED
7-78
DDC

```
C     * INPUT - NONE OUTPUT - Z   *
C     *                           *
C     *****************************
C
      READ (1,601) (Z(I),I=1,3)
601   FORMAT (3F13.4)
C
C     *****************************
C     *                           *
C     *    PROPAGATE MODULE       *
C     *                           *
C     * INPUT - X    OUTPUT - X   *
C     *       - P           - P   *
C     *       - Q                 *
C     *       - PHI               *
C     *                           *
C     *****************************
C
C
C     PROPAGATE X VECTOR
C
      DO 110 I=1,7,3
      J = I + 1
      K = I + 2
      X(I) = X(I) + PHI12 * X(J) + PHI13 * X(K)
      X(J) = X(J) + PHI23 * X(K)
      X(K) = PHI33 * X(K)
110   CONTINUE
C
C     PROPAGATE P MATRIX
```

```
C
      DO 115 I=1,7,3
      J = I + 1
      K = I + 2
      P(J,I) = P(I,J)
      P(K,I) = P(I,K)
      P(K,J) = P(J,K)
115   CONTINUE
C
      DO 120 IA=1,7,3
      IB = IA + 1
      IC = IA + 2
      DO 120 I = IA,7,3
      J = I + 1
      K = I + 2
      A(1) = P(IA,I)
      A(2) = P(IB,I)
      A(3) = P(IC,I)
      A(4) = P(IA,J)
      A(5) = P(IB,J)
      A(6) = P(IC,J)
      A(7) = P(IA,K)
      A(8) = P(IB,K)
      A(9) = P(IC,K)
      P(IA,I)=A(1)+PHI12*A(2)+PHI13*A(3)+PHI12*(A(4)+PHI12*A(5)+PHI13*
     CA(6))+PHI13*( A(7)+PHI12*A(8)+PHI13*A(9))
      P(IA,J)=A(4)+PHI12*A(5)+PHI13*A(6)+PHI23*(A(7)+PHI12*A(8)+PHI13*
     CA(9))
      P(IA,K)=PHI33*(A(7)+PHI12*A(8)+PHI13*A(9))
      P(IB,J)=A(5)+PHI23*(A(6)+A(8)+PHI23*A(9))
```

```
      P(I8,K)=PHI33*(A(8)+PHI23*A(9))
      P(IC,K)=PHI33*PHI33*A(9)
      IF ( IA .EQ. I ) GO TO 120
      P(I9,I)=A(2)+PHI23*A(3)+PHI12*(A(5)+PHI23*A(6))+PHI13*(A(8)+PHI23*
     CA(9))
      P(IC,I)=PHI33*(A(3)+PHI12*A(6)+PHI13*A(9))
      P(IC,J)=PHI33*(A(6)+PHI23*A(9))
120   CONTINUE
C
      DO 140 I=1,7,3
      J = I + 1
      K = I + 2
      P(I,I) = P(I,I) + QX
      P(J,J) = P(J,J) + QV
      P(K,K) = P(K,K) + QA
140   CONTINUE
C
C     ********************************
C     *                              *
C     *       UPDATE MODULE          *
C     *                              *
C     * INPUT - Z    OUTPUT - X      *
C     *       - X             - P    *
C     *       - P                    *
C     *       - R                    *
C     *                              *
C     ********************************
C
C
C     UPPER TRIANGULAR CHOLESKY FACTORIZATION
```

```
C
      DO 17 I=1,9
      DO 17 J=I,9
      S(I,J) = 0.0
17    CONTINUE
C
      DO 25 I2 = 2,9
      J = 11 - I2
      S(J,J) = SQRT (P(J,J))
      ALPHA = 1. / S(J,J)
      L = J - 1
      DO 25 K=1,L
      S(K,J) = ALPHA * P(K,J)
      BETA = S(K,J)
      DO 25 I=1,K
      P(I,K) = P(I,K) - BETA * S(I,J)
25    CONTINUE
      S(1,1) = SQRT (P(1,1))
C
      DO 50 L=1,3
C
      DO 290 I=2,9
      A(I) = 0.0
290   CONTINUE
C
C     COMPUTE ZP AND H
C
      IF ( L .NE. 1 ) GO TO 210
      TEMP   = SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      ZP(1) = TEMP
```

```
      TEMP = 1. / TEMP
      A(1) = X(1) * TEMP
      A(4) = X(4) * TEMP
      A(7) = X(7) * TEMP
      GO TO 230
210   IF ( L .NE. 2 ) GO TO 220
      UA = X(4) / X(1)
      TEMP = 1. / ((1. + UA * UA) * X(1))
      ZP(2) = ATAN(UA)
      A(1) = - UA * TEMP
      A(4) = TEMP
      GO TO 230
220   TEMP = 1. / SQRT(X(1) * X(1) + X(4) * X(4) + X(7) * X(7))
      VA = X(7) * TEMP
      DENOMC = SQRT(1. - VA * VA)
      ZP(3) = ASIN(VA)
      A(1) = - VA * X(1) * TEMP * TEMP / DENOMC
      A(4) = - VA * X(4) * TEMP * TEMP / DENOMC
      A(7) = DENOMC * TEMP
C
C     UPDATE U,X
C
230   Z(L) = Z(L) - ZP(L)
      DO 10 I2 = 1,9
      J = 10 - I2
      SIGMA = 0.
      DO 5 K=1,J,3
      SIGMA = SIGMA + S(K,J) * A(K)
5     CONTINUE
      A(J) = SIGMA
```

```
10      CONTINUE
C
        ALPHA = R(L) + A(1) * A(1)
        B(1) = S(1,1) * A(1)
        S(1,1) = S(1,1) * SQRT (R(L) / ALPHA)
        DO 20 J=2,9
        B(J) = S(J,J) * A(J)
        GAMMA = ALPHA
        ALPHA = ALPHA + A(J) * A(J)
        BETA = SQRT (GAMMA / ALPHA)
        GAMMA = A(J) / (ALPHA * BETA)
        S(J,J) = S(J,J) * BETA
        I1 = J - 1
        DO 20 K=1,I1
        T = S(K,J)
        S(K,J) = T * BETA - GAMMA * B(K)
        B(K) = B(K) + A(J) * T
20      CONTINUE
        Z(L) = Z(L) / ALPHA
        DO 30 J=1,9
        X(J) = X(J) + B(J) * Z(L)
30      CONTINUE
50      CONTINUE
C
C       CONVERT P
C
        DO 300 I=1,9
        DO 300 J=1,I
        TEMP = 0.0
        DO 350 K=I,9
```

```
      TEMP = TEMP + S(I,K) * S(J,K)
350   CONTINUE
      P(J,I) = TEMP
300   CONTINUE
C
C     ******************************
C     *                            *
C     *         WRITE MODULE       *
C     *                            *
C     * INPUT - X    OUTPUT - NONE *
C     *                            *
C     ******************************
C
      WRITE (2,602) (X(I),I=1,9)
602   FORMAT ( F20.15 )
900   CONTINUE
      STOP
      END
```

Appendix C.6

Filter Six: Standard Kalman Filter Program

Filter six represents the usual method of implementing the extended Kalman filter into a computer program. It processes the measurement vector all at once. This is accomplished through the use of general purpose subroutines that waste storage and require additional time to execute. The program employs the covariance update technique of

$$\bar{P}(t_i^+) = \bar{P}(t_i^-) - \bar{\bar{K}}\bar{\bar{H}}\bar{\bar{P}}(t_i^-) \qquad (35)$$

Since this program is the usual method of implementing the extended Kalman filter, it serves as a worst case reference for the other filters.

This program is written in American National Standards Institute (ANSI) FORTRAN, as described in X.3-1966 with the exception of the PROGRAM instruction. Program documentation consists of internal comment cards, Fig. 10, and the usual Kalman filter equations (Ref 6:65-66).

```
      PROGRAM EXN05(INPUT,OUTPUT,TAPE1=INPUT,TAPE2=OUTPUT)
      DIMENSION PHI(9,9),Q(9,9),P(9,9),X(9,1),R(3,3),H(3,9),Z(3,1),
     C Y(9,3),  W(9,9),V(9,9),G(9,3),ZP(3,1),F(9,1),S(3,3),T(3,3),U(3,9)
      REAL DELTA,DENOMC,DENOMB,DENOMA,F,G,H,P,PHI,Q,R,S,T,TAU,TEMP,U,UA,
     C V,VA,W,X,Y,Z,ZP
      INTEGER I,J,K,M
C
CCC   ********************************
CCC   *                              *
CCC   *     INITIATE MODULE          *
CCC   *                              *
CCC   * INPUT - NONE OUTPUT - X      *
CCC   *                     - P      *
CCC   *                     - GQGT   *
CCC   *                     - PHI    *
CCC   *                     - R      *
CCC   *                              *
CCC   ********************************
C
      DELTA = .1
      TAU = 10.
      TEMP = EXP( - DELTA / TAU )
      DO 11 I=1,9
      DO 11 J=1,9
      PHI(I,J) = 0.0
11    CONTINUE
      DO 12 I=1,7,3
      J = I + 1
      K = I + 2
      PHI(I,I) = 1.0
```

```
      PHI(I,J) = DELTA
      PHI(I,K) = ( -TAU + DELTA + TAU * TEMP ) * TAU
      PHI(J,J) = 1.0
      PHI(J,K) = TAU * (1. - TEMP)
      PHI(K,K) = TEMP
12    CONTINUE
      DO 13 I=1,9
      DO 13 J=1,9
      Q(I,J) = 0.0
13    CONTINUE
      DO 14 I=1,7,3
      J = I + 1
      K = I + 2
      Q(I,I) = .0001
      Q(J,J) = .00001
      Q(K,K) = .000001
14    CONTINUE
      DO 16 I=1,9
      DO 16 J=1,9
      P(I,J) = 0.0
16    CONTINUE
      P(1,1) = 1.
      P(2,2) = .01
      P(3,3) = .001
      P(4,4) = 1.
      P(5,5) = .01
      P(6,6) = .001
      P(7,7) = 1.
      P(8,8) = .01
      P(9,9) = .001
```

```
      X(1,1) = 54.338
      X(2,1) = 0.0
      X(3,1) = 0.0
      X(4,1) = 0.0
      X(5,1) = 0.0
      X(6,1) = 0.0
      X(7,1) = 0.0
      X(8,1) = 0.0
      X(9,1) = 0.0
      DO 17 I=1,3
      DO 17 J=1,3
      R(I,J) = 0.0
17    CONTINUE
      R(1,1) = .0001
      R(2,2) = .00001
      R(3,3) = .00001
      DO 18 I=1,3
      DO 18 J=1,9
      P(I,J) = 0.0
18    CONTINUE
C
      DO 900 M=1,100
C
C     ********************************
C     *                              *
C     *         READ MODULE          *
C     *                              *
C     * INPUT - NONE OUTPUT - Z      *
C     *                              *
C     ********************************
```

```
C
      READ (1,601) (Z(I,1),I=1,3)
601   FORMAT (3F13.4)
C
C     ********************************
C     *                              *
C     *    PROPAGATE MODULE          *
C     *                              *
C     * INPUT - X     OUTPUT - X     *
C     *       - P              - P   *
C     *       - GQGT                 *
C     *       - PHI                  *
C     *                              *
C     ********************************
C
C
C     PROPAGATE X VECTOR
C
      CALL MMUL(PHI,X,F,9,9,1)
      CALL MMOV(F,X,9,1)
C
C     PROPAGATE P MATRIX
C
      CALL MTRA(PHI,W,9,9)
      CALL MMUL(P,W,V,9,9,9)
      CALL MMUL(PHI,V,P,9,9,9)
      CALL MADD(P,Q,P,9,9)
C
C     ********************************
C     *                              *
```

```
C     *        UPDATE MODULE        *
C     *                             *
C     * INPUT - Z    OUTPUT - X     *
C     *       - X             - P   *
C     *       - P                   *
C     *       - R                   *
C     *                             *
C     *******************************
C
      DENOMA = SQRT(X(1,1) * X(1,1) + X(4,1) * X(4,1) + X(7,1) * X(7,1))
      UA = X(4,1) / X(1,1)
      VA = X(7,1) / DENOMA
      DENOMB = (1. + UA * UA) * X(1,1)
      DENOMC = SQRT(1. - VA * VA)
C
C     COMPUTE ZP VECTOR AND H MATRIX
C
      ZP(1,1) = DENOMA
      ZP(2,1) = ATAN(UA)
      ZP(3,1) = ASIN(VA)
      H(1,1) = X(1,1) / DENOMA
      H(1,4) = X(4,1) / DENOMA
      H(1,7) = X(7,1) / DENOMA
      H(2,1) = - UA / DENOMB
      H(2,4) = 1. / DENOMB
      H(3,1) = - VA * X(1,1) / (DENOMC * DENOMA * DENOMA)
      H(3,4) = - VA * X(4,1) / (DENOMC * DENOMA * DENOMA)
      H(3,7) = DENOMC / DENOMA
C
C     UPDATE G MATRIX
```

```
C
      CALL MTRA(H,Y,3,9)
      CALL MMUL(P,Y,G,9,9,3)
      CALL MMUL(H,G,S,3,9,3)
      CALL MADD(S,R,S,3,3)
      CALL MINV(S,T,3)
      CALL MTRA(H,G,3,9)
      CALL MMUL(G,T,Y,9,3,3)
      CALL MMUL(P,Y,G,9,9,3)
C
CC    UPDATE P MATRIX
C
      CALL MMUL(H,P,U,3,9,9)
      CALL MMUL(G,U,V,9,3,9)
      CALL MSBT(P,V,P,9,9)
C
CC    UPDATE X VECTOR
C
      CALL MSBT(Z,ZP,Z,3,1)
      CALL MMUL(G,Z,F,9,3,1)
      CALL MADD(X,F,X,9,1)
C
CC    ********************************
CC    *                              *
CC    *         WRITE MODULE         *
CC    *                              *
CC    * INPUT - X    OUTPUT - NONE   *
CC    *                              *
CC    ********************************
C
```

```
      WRITE (2,602) (X(I,1),I=1,9)
602   FORMAT ( F20.15 )
900   CONTINUE
      STOP
      END
C
      SUBROUTINE MADD(A,B,C,M,N)
      DIMENSION A(M,N),B(M,N),C(M,N)
      REAL A,B,C
      INTEGER M,N
      DO 10 I=1,M
      DO 10 J=1,N
      C(I,J) = A(I,J) + B(I,J)
10    CONTINUE
      RETURN
      END
C
      SUBROUTINE MSBT(A,B,C,M,N)
      DIMENSION A(M,N),B(M,N),C(M,N)
      REAL A,B,C
      INTEGER M,N
      DO 20 I=1,M
      DO 20 J=1,N
      C(I,J) = A(I,J) - B(I,J)
20    CONTINUE
      RETURN
      END
C
      SUBROUTINE MTRA(A,B,M,N)
      DIMENSION A(M,N),B(N,M)
```

```
      REAL A,B
      INTEGER M,N
      DO 40 I=1,M
      DO 40 J=1,N
      B(J,I) = A(I,J)
40    CONTINUE
      RETURN
      END
C
      SUBROUTINE MMOV(A,B,M,N)
      DIMENSION A(M,N),B(M,N)
      REAL A,B
      INTEGER M,N
      DO 50 I=1,M
      DO 50 J=1,N
      B(I,J) = A(I,J)
50    CONTINUE
      RETURN
      END
C
      SUBROUTINE MMUL(A,B,C,L,M,N)
      DIMENSION A(L,M),B(M,N),C(L,N)
      REAL A,B,C
      INTEGER L,M,N
      DO 30 I=1,L
      DO 30 J=1,N
      TEMP = 0.0
      DO 31 K=1,M
      TEMP = TEMP + A(I,K) * B(K,J)
31    CONTINUE
```

```
      C(I,J) = TEMP
30    CONTINUE
      RETURN
      END
C
      SUBROUTINE MINV(A,B,M)
      DIMENSION A(M,M),B(M,M)
      REAL A,B
      INTEGER M
      DET = A(1,1)*(A(2,2)*A(3,3)-A(3,2)*A(2,3))-
     C      A(2,1)*(A(1,2)*A(3,3)-A(3,2)*A(1,3))+
     C      A(3,1)*(A(1,2)*A(2,3)-A(2,2)*A(1,3))
      B(1,1) =   (A(2,2) * A(3,3) - A(3,2) * A(2,3)) / DET
      B(1,2) = - (A(1,2) * A(3,3) - A(3,2) * A(1,3)) / DET
      B(1,3) =   (A(1,2) * A(2,3) - A(2,2) * A(1,3)) / DET
      B(2,1) = - (A(2,1) * A(3,3) - A(3,1) * A(2,3)) / DET
      B(2,2) =   (A(1,1) * A(3,3) - A(3,1) * A(1,3)) / DET
      B(2,3) = - (A(1,1) * A(2,3) - A(2,1) * A(1,3)) / DET
      B(3,1) =   (A(2,1) * A(3,2) - A(3,1) * A(2,2)) / DET
      B(3,2) = - (A(1,1) * A(3,2) - A(3,1) * A(1,2)) / DET
      B(3,3) =   (A(1,1) * A(2,2) - A(2,1) * A(1,2)) / DET
      RETURN
      END
```

Vita

Henry Joseph Laughlin was born 29 March 1948 in Latrobe, Pennsylvania. He graduated from the Derry Area Senior High School, Derry, Pennsylvania in 1967. After high school he attended the United States Air Force Academy and graduated in 1971.

His military career has included experience in TITAN II ICBM Operations. He has held positions from line crew deputy to Alternate Command Post Commander.

In 1976 he was accepted by AFIT for the Graduate Computer Systems Program.

Permanent address:

Capt. Henry J. Laughlin
R. D. #5 Box 112
Latrobe, PA 15650

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/78-3	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) INVESTIGATION OF REAL-TIME SATELLITE-TO-SATELLITE TRACKING USING EXTENDED KALMAN FILTERS		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Henry J. Laughlin Capt USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, OHIO 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS SAMSO/YAS LAAFS, PO BOX 92960 Worldway Postal Center Los Angeles, Ca. 90009		12. REPORT DATE March, 1978 13. NUMBER OF PAGES 103
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release: IAW AFR 190-17

Jerral F. Guess, Captain USAF
Director of Information

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Extended Kalman Filter
Tracking Problem
Real-time Application

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The feasibility of adapting an extended Kalman filter to the real-time satellite-to-satellite tracking problem is investigated. A filter using linear propagation and nonlinear measurements is formulated. This filter accepts tracker measurements to provide the state estimates (position, velocity, and acceleration) of the tracked satellite in the local inertial reference frame.

Six different filters are programmed in American National Standards Institute (ANSI) FORTRAN. These filters employ (cont)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 Cont. various covariance update techniques and methods of processing the measurement vector. The programs are compared for computer program storage requirements, program execution speeds, and computational precision.

The computer program storage requirements are deter mined by utilizing the FORTRAN IV-PLUS compiler on a PDP-11/45 minicomputer. The program execution speeds are found by timing on the PDP-11/45 the filter's reaction to a typical rendezvous between two space vehicles. The computer precision of each filter is determined by comparing the state estimates obtained from the 60-bit word CDC-6600 computer with those from the 32-bit word PDP-11/45 minicomputer. A final filter is selected as best for the real-time satellite-to-satellite tracking problem.